Program Slicing

Mark Harman,
Department of Mathematical and Computing Sciences,
Goldsmiths College, University of London,
Lewisham Way, New Cross,
London SE14 6NW, UK.
Tel: +44 (0)171 919 7860.
Fax: +44 (0)171 919 7853.
M.Harman@gold.ac.uk

Keith Brian Gallagher,
Computer Science Department,
Loyola College in Maryland,
4501 North Charles Street,
Baltimore, Maryland 21210–2699, USA.
Tel: +1 410 617 2854
Fax: +1 410 617 2157
kbg@loyola.edu

1 Introduction

Many software technology problems can be characterised as problems of scale, one of the most important of which is the shear size of the source code which describes the system itself. Program slicing addresses this problem head on, reducing program size by focusing attention on a sub computation denoted by a 'slicing criterion'.

Mark Weiser [51] introduced program slicing in 1979. Weiser's thesis inspired much theoretical and practical development, which continues to the present day. The slicing criterion introduced by Weiser was a pair (V, n), containing a set of variables, V and a program point n. Statements and predicates[1] in the original program which cannot affect any variable in V when the next statement to be executed is at n, are removed to form a slice constructed with respect to (V, n). Figure 1 depicts a program fragment and one of its slices, constructed with respect to the slicing criterion $(\{\mathtt{s}\}, 9)$.

Research in program slicing can be divided into three broad categories:

1. **Applications**.
   Slicing identifies a subcomputation from a program, thereby providing a wide and varied application set.
2. **Criteria**.
   Slicing criteria form the basis for a number of extensions to the basic principle of slicing.
3. **Algorithms**.
   Much slicing work concerns improvements and extensions to algorithms for slice construction.

This special issue contains papers concerned with all three of these categories. However, it naturally only covers a small sample of current work on program slicing. Surveys can be found in the work of Binkley and Gallagher [7] and Tip [48].

2 Applications of Slicing

Slicing can be used as an assistant in many aspects of the software development life cycle. This section lists some of these applications, paying particular attention to those described in the papers which follow.

[1] Strictly speaking, it is nodes of the program's Control Flow Graph which are considered for removal.

1	`scanf("%d",&n);`	1	`scanf("%d",&n);`
2	`s=0;`	2	`s=0;`
3	`p=1;`		
4	`while (n>0)`	4	`while (n>0)`
5	`{ s=s+n;`	5	`{ s=s+n;`
6	`p=p*n;`		
7	`n=n-1;`	7	`n=n-1; }`
8	`printf("%d",s);`		
9	`printf("%d",p); }`		
Original program fragment		Slice with respect to $(\{\mathtt{s}\}, 9)$	

Figure 1: A Program Fragment and One of its Slices

2.1 Debugging

Slicing was first applied to the problem of debugging [52, 39]. If a bug manifests itself as an incorrect value in a variable, then slicing can be used to help track down the statement or statements which cause the incorrect value, reducing the amount of code that the human debugger must consider.

More recently, slicing has been applied to the problem of algorithmic debugging [46, 28]. That is, the (semi) automated the task of homing in on the program sites which cause an error. Mariam Kamkar's paper describes this application in more detail.

2.2 Maintenance and Regression Testing

Bug detection normally leads to bug correction. The consequent change (software surgery) may well produce further bugs, because of the unintentional introduction of ripple effects. Gallagher and Lyle [18] showed how slicing can be used to assess the extent of these ripple effects, thereby controlling the process of software surgery.

Having made a series of changes, the system is typically re-tested with all its original input in a 'regression testing' phase. Clearly there is no point in re-testing parts of the software which could not have been affected by any of the changes made. Slicing therefore has a role to play in reducing testing effort [20, 22]. This application is discussed in more detail in David Binkley's paper.

2.3 Reuse, Re-structuring and Re-engineering

Some systems are beyond repair. In such a situation, a re-engineering process needs to be undertaken if the 'software asset' is to be salvaged. Even where systems are not beyond repair, it is often useful to be able to extract components of the systems for reuse and re-structuring. Slicing has been used as a component in strategies for restructuring [34, 47], re-engineering [13, 8], comprehension [15, 23, 27] and reuse [5, 12, 11]. Many of these applications are described in the paper by Gerardo Canfora, Andrea DeLucia and Aniello Cimitile and the paper by Arun Lakhotia and Jean–Christophe Deprez.

2.4 Measurement

Software measurement is applicable at several stages of the development life cycle. Unfortunately, many of the most interesting and important software attributes have proved resistant to measurement. Cohesion is an example of just such an attribute. It is often informally defined in the form of seven cohesion 'levels' [14].

Slicing has been applied to the problem of formalising cohesion measurement. Bieman and Ott used the intersection of a program's slices as a way of locating and measuring the cohesive portions of the code [6, 42, 41, 43, 40]. Lakhotia used the dependence analysis upon which slicing is based to formalise the seven informally defined levels of cohesion [32]. Work on slice-based cohesion measurement is described in more detail in the paper by Jim Bieman and Linda Ott.

2.5 Validation

Several large scale program slicers have been built, for example the Wisconsin Program Slicing Tool [24] and the Unravel Slicing Tool [38]. Slicing also formed the basis of a project involving the Technical University of Braunschweig, the PTB (Physikalisch-Technische Bundesanstalt — the national metrology institute), and the software company LINEAS. This project, VALSOFT, was concerned with the validation of control software used in physical measurement systems, such as weighing scales. The goal was to identify paths which exhibit suspicious information flows.

The VALSOFT project suggested that it would be useful to combine traditional slicing technology with constraint satisfaction problems, yielding more precise slices. The VALSOFT project and the connection between slicing and constraint satisfaction are described in the paper by Jens Krinke and Gregor Snelting.

3 Forms Of Slicing

Mark Weiser's original slicing criterion is known as the 'static backward' slicing criterion [50]. It is 'static' in the sense that any statement which is deleted can have no effect upon the slicing criterion when the program is executed in *any* initial state — input conditions are not considered. It is 'backward' because, in constructing the slice, the control flow of the program is considered 'in reverse', to ascertain the statements and predicates which can affect the slicing criterion. Slices can be also constructed in a 'forward' direction [26] and with respect to a variety of non-static criteria.

The reader will have noticed that the example in Figure 1 has been carefully constructed to contain two distinct computations which have been 'married together' in a single loop. This is somewhat artificial, atypical and of illustrative value only. Code is often found to be highly cohesive with the result that static slicing on many interesting criteria leads to unhelpfully large slices. This observation was one factor which pushed slicing research in the direction of richer slicing criteria, such as the dynamic and conditioned criteria.

3.1 Dynamic Slicing

A statement which is removed in static slice construction can have no effect upon the subcomponent identified by the slicing criterion in *any* execution of the system. On the other hand, a dynamic slice [30, 3] is constructed with respect to an augmented slicing criterion which (additionally) contains the sequence of input supplied to the program for a particular execution. This additional information makes dynamic slices far smaller than their static counterparts, but the information presented in the slice is only valid for a single program execution. Dynamic slicing is thus more useful than static slicing for testing and debugging, but less useful for comprehension and reuse.

There are two forms of dynamic slice. Korel and Laski [30] introduced the first formulation, which requires that the slice preserve the effect of the original program on V at n when the program is executed with the input sequence I. The dynamic slice constructed according to this formulation is an executable program, which preserves the effect of the original upon the set of variables in V when the input is I and the next statement to be executed is at node n. Subsequently, Agrawal and Horgan [3], introduced a form of dynamic slice which is not executable. Rather it consists simply of the set of statements and predicates which can affect V at n on I.

Dynamic slicing is discussed in more detail in the paper by Bogdan Korel and Jurgen Rilling.

3.2 Conditioned Slicing

Conditioned (or constrained [17]) slicing forms a 'bridge' between dynamic and static slicing. A conditioned slice is constructed with respect to a set of initial states and a static slicing criterion. The set of initial states can be singleton (essentially yielding dynamic slicing) or it can be the set of all states (yielding static slicing). Conditioned slicing is discussed in more detail in the paper by Gerardo Canfora, Andrea DeLucia and Aniello Cimitile.

4 Algorithms for Slicing

The definition of a program slice seems so straightforward and intuitive that is is tempting to think that automated slice construction would be an equally straightforward and consequently undemanding task. This is a misconception. Minimal slices are neither computable nor unique [2] [51]. In order to construct a slice, a slicing tool requires relatively sophisticated semantic information regarding inter-dependences between programming language constructs. This requirement has forced slicing research to address problems associated with difficult programming language features, such as procedural abstraction [26, 45], pointers [2, 35, 37, 16], `goto` statements [21, 29, 10, 1, 4], object orientation [33, 49] and concurrent execution [31, 9, 53].

Classical approaches [52, 36, 19] rely upon a collection of data flow equations relating a Control Flow Graph (CFG) node to the variable names which are 'relevant' (with respect to the slicing criterion) at that node. The slice is the least solution to this set of equations. In 1984, Ottenstein and Ottenstein [44], showed that slices could be constructed from a form of Program Dependence Graph (PDG), whose nodes are essentially those of the CFG and whose edges represent control and data dependence between nodes. This reduced the problem of slice construction to the problem of PDG construction, followed by a simple traversal of PDG edges. In 1988, Horwitz, Reps and Binkley [25] extended the PDG to the System Dependence Graph (SDG), a richer dependence graph capable of handling interprocedural slicing, while accurately accounting for procedure calling contexts.

The paper by Thomas Reps describes a recent development in interprocedural slicing, which offers improved time-complexity over the original SDG construction algorithm. The new algorithm relies upon the reformulation of interprocedural analysis as a problem in Context Free Language Reachability (CFLR). As indicated in Reps' paper, the result is important not only because it provides an improved interprocedural slicing algorithm, but also because it opens the way for a number of similar programming language analysis problems to be reformulated as CFLR problems.

The paper by John Field and Frank Tip describes a novel approach to slice construction, based upon a term rewriting system called PIM. The intermediate language used by Field and Tip, allows them to represent, not only the computation performed by a program, but also a projection of both its initial and final state. This, in turn, allows for the definition of a set of powerful directed transformation rules, which achieve simplification when the state projection is non-trivial. The PIM system maintains a link between terms in expressions and terms in rewritten expressions, allowing new terms to be related to old ones. This additional 'origin tracking' makes it possible to apply the PIM approach to program slicing problems, where the identification of deleted syntactic components is important.

5 This Issue of Information and Software Technology

This special issue of Information and Software Technology contains nine papers which describe some of the forms of, algorithms for and applications of program slicing.

Sadly, space has not allowed the guest editors to include all the exciting developments in this field, but it is hoped that this issue contains sufficient of the state of the art to provide the reader with a feeling for the vibrancy and diversity of slicing research.

The editors would like to extend their heartfelt thanks to all those who have made this special issue possible. The authors of the nine papers and to Mark Weiser for his foreword, Doutzen Alma at Elsevier and Jackie Holmes at Bournemouth University, for co-ordinating the production of the issue and to Martin Shepperd who originally put forward the idea of a special issue on Program Slicing, and who has remained interested and supportive throughout.

Mark Harman,	Keith Brian Gallagher,
London, UK.	Maryland, USA.

[2] although it is always possible to define computable, but conservative, approximations which are safe in the sense that the computed slices *contain* the *real* slices as subsets.

References

[1] Hiralal Agrawal. On slicing programs with jump statements. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 302–312, Orlando, Florida, June 20–24 1994. Proceedings in SIGPLAN Notices, 29(6), June 1994.

[2] Hiralal Agrawal, Richard A. DeMillo, and Eugene H. Spafford. Dynamic slicing in the presence of unconstrained pointers. In 4^{th} *ACM Symposium on Testing, Analysis, and Verification (TAV4)*, pages 60–73, 1991. Appears as Purdue University Technical Report SERC-TR-93-P.

[3] Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 246–256, New York, June 1990.

[4] Thomas Ball and Susan Horwitz. Slicing programs with arbitrary control–flow. In Peter Fritzson, editor, 1^{st} *Conference on Automated Algorithmic Debugging*, pages 206–222. Springer, 1993. Also available as Uniersity of Wisconsin–Madison, technical report (in extended form), TR-1128, December, 1992.

[5] Jon Beck and David Eichmann. Program and interface slicing for reverse engineering. In *IEEE/ACM* 15^{th} *Conference on Software Engineering (ICSE'93)*, pages 509–518, 1993.

[6] James M. Bieman and Linda M. Ott. Measuring functional cohesion. *IEEE Transactions on Software Engineering*, 20(8):644–657, August 1994.

[7] David W. Binkley and Keith Brian Gallagher. Program slicing. In Marvin Zelkowitz, editor, *Advances of Computing, Volume 43*. Academic Press, 1996.

[8] Gerardo Canfora, Aniello Cimitile, Andrea De Lucia, and G. A. Di Lucca. Software salvaging based on conditions. In *International Conference on Software Maintenance (ICSM'96)*, pages 424–433, Victoria, Canada, September 1994. IEEE.

[9] Jingde Cheng. Slicing concurrent programs – a graph–theoretical approach. In Peter Fritzson, editor, 1^{st} *Automatic Algorithmic Debugging Conference (AADEGUB'93)*, pages 223–240, 1993. Appears as Springer Lecture Notes in Computer Science vol 749.

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

[11] Aniello Cimitile, Andrea De Lucia, and Malcolm Munro. Identifying reusable functions using specification driven program slicing: a case study. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'95)*, pages 124–133, Nice, France, 1995.

[12] Aniello Cimitile, Andrea De Lucia, and Malcolm Munro. Qualifying reusable functions using symbolic execution. In *Proceedings of the* 2^{nd} *working conference on reverse engineering*, pages 178–187, Toronto, Canada, 1995. IEEE Press.

[13] Aniello Cimitile, Andrea De Lucia, and Malcolm Munro. A specification driven slicing process for identifying reusable functions. *Software maintenance: Research and Practice*, 8:145–178, 1996.

[14] Larry L. Constantine and Edward Yourdon. *Structured Design*. Prentice Hall, 1979.

[15] Andrea De Lucia, Anna Rita Fasolino, and Malcolm Munro. Understanding function behaviours through program slicing. In 4^{th} *IEEE Workshop on Program Comprehension*, Berlin, Germany, March 1996.

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

[17] John Field, G. Ramalingam, and Frank Tip. Parametric program slicing. In 22^{nd} *ACM Symposium on Principles of Programming Languages*, pages 379–392, San Francisco, CA, 1995.

[18] Keith B. Gallagher and James R. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, August 1991.

[19] Keith Brian Gallagher. *Using program slicing in software maintenance*. PhD thesis, University of Maryland, College Park, Maryland, January 1990.

[20] Rajiv Gupta, Mary Jean Harrold, and Mary Lou Soffa. An approach to regression testing using slicing. In *Proceedings of the IEEE Conference on Software Maintenance*, pages 299–308, Orlando, Florida, USA, 1992. IEEE Computer Society Press.

[21] Mark Harman and Sebastian Danicic. A new algorithm for slicing unstructured programs. *Journal of Software Maintenance*, page to appear.

[22] Mark Harman and Sebastian Danicic. Using program slicing to simplify testing. *Journal of Software Testing, Verification and Reliability*, 5:143–162, September 1995.

[23] Mark Harman and Sebastian Danicic. Amorphous program slicing. In 5^{th} *IEEE International Workshop on Program Comprehesion (IWPC'97)*, pages 70–79, Dearborn, Michigan, USA, May 1997.

[24] Susan Horwitz and Thomas Reps *et al.* Wisonsin program slicing project. URL `http://www.cs.wisc.edu/wpis/html/`.

[25] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 25–46, Atlanta, Georgia, June 1988. Proceedings in *SIGPLAN Notices*, 23(7), pp.35–46, 1988.

[26] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12:26–61, 1990.

[27] Daniel Jackson and Eugene J. Rollins. Chopping: A generalisation of slicing. Technical Report CMU-CS-94-169, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 1994.

[28] Mariam Kamkar. *Interprocedural dynamic slicing with applications to debugging and testing.* PhD Thesis, Department of Computer Science and Information Science, Linköping University, Sweden, 1993. Available as Linköping Studies in Science and Technology, Dissertations, Number 297.

[29] Bogdan Korel. Computation of dynamic slices for programs with arbitrary control flow. In Mireille Ducassé, editor, 2^{nd} *International Workshop on Automated Algorithmic Debugging (AADEBUG'95)*, Saint–Malo, France, May 1995.

[30] Bogdan Korel and Janusz Laski. Dynamic program slicing. *Information Processing Letters*, 29(3):155–163, October 1988.

[31] Jens Krinke. Static slicing of threaded programs. In *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'98)*, pages 35–42, Montreal, Canada, June 1998.

[32] Arun Lakhotia. Rule–based approach to computing module cohesion. In *Proceedings of the* 15^{th} *Conference on Software Engineering (ICSE-15)*, pages 34–44, 1993.

[33] Loren D. Larsen and Mary Jean Harrold. Slicing object-oriented software. In *Proceedings of the 18th International Conference on Software Engineering*, pages 495–505, Berlin, 1996.

[34] Lulu Liu and Rod Ellis. An approach to eliminating COMMON blocks and deriving ADTs from Fortran programs. Technical report, University of Westminster, UK, February 1993.

[35] 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.

[36] James R. Lyle. *Evaluating Variations of Program Slicing for Debugging.* PhD thesis, University of Maryland, College Park, Maryland, December 1984.

[37] James R. Lyle and David Binkley. Program slicing in the presence of pointers. In *Foundations of Software Engineering*, 1993.

[38] James R. Lyle, Dolores R. Wallace, James R. Graham, Keith B. Gallagher, Joseph P. Poole, and David W. Binkley. Unravel project. URL `http://hissa.ncsl.nist.gov/~jimmy/unravel.html`.

[39] James R. Lyle and Mark Weiser. Automatic program bug location by program slicing. In 2^{nd} *International Conference on Computers and Applications*, pages 877–882, Peking, 1987.

[40] Linda M. Ott. Using slice profiles and metrics during software maintenance. In *Proceedings of the* 10^{th} *Annual Software Reliability Symposium*, pages 16–23, 1992.

[41] Linda M. Ott and James M. Bieman. Effects of software changes on module cohesion. In *IEEE Conference on Software Maintenance*, pages 345–353, November 1992.

[42] Linda M. Ott and J. J. Thuss. The relationship between slices and module cohesion. In *Proceedings of the* 11^{th} *ACM conference on Software Engineering*, pages 198–204, May 1989.

[43] Linda M. Ott and Jeff J. Thuss. Slice based metrics for estimating cohesion. In *Proceedings of the IEEE-CS International Metrics Symposium*, pages 78–81, 1993.

[44] Karl J. Ottenstein and Linda M. Ottenstein. The program dependence graph in software development environments. *SIGPLAN Notices*, 19(5):177–184, 1984.

[45] Thomas Reps, Susan Horwitz, Mooly Sagiv, and Genevieve Rosay. Speeding up slicing. In *ACM Foundations of Software Engineering (FSE'94)*, pages 11–20, New Oleans, LA, December 1994.

[46] Nahid Shahmehri. *Generalized algorithmic debugging.* PhD Thesis, Department of Computer Science and Information Science, Linköping University, Sweden, 1991. Available as Linköping Studies in Science and Technology, Dissertations, Number 260.

[47] Dan Simpson, Samuel H. Valentine, Richard Mitchell, Lulu Liu, and Rod Ellis. Recoup – Maintaining Fortran. *ACM Fortran forum*, 12(3):26–32, September 1993.

[48] Frank Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, September 1995.

[49] Frank Tip, Jong-Deok Choi, John Field, and G. Ramalingham. Slicing class hierarchies in C++. In *Proceedings of the* 11^{th} *conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'96)*, San Jose, October 1996.

[50] Guda A. Venkatesh. The semantic approach to program slicing. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 26–28, Toronto, Canada, June 1991. Proceedings in *SIGPLAN Notices*, 26(6), pp.107–119, 1991.

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

[52] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.

[53] Jianjun Zhao, Jingde Cheng, and Kazuo Ushijima. Static slicing of concurrent object-oriented programs. In 20^{th} *IEEE Annual International Computer Software and Applications Conference (COMPSAC'96)*, Seoul, Korea, August 1996. To appear.