05451 Abstracts Collection
Beyond Program Slicing
— Dagstuhl Seminar —

Dave Binkley[1], Mark Harman[2] and Jens Krinke[3]

[1] Loyola College - Baltimore, US
binkley@cs.loyola.edu
[2] King's College London, GB
Mark.Harman@kcl.ac.uk
[3] FernUniversität in Hagen, D
krinke@acm.org

Abstract. From 06.11.05 to 11.11.05, the Dagstuhl Seminar 05451 "Beyond Program Slicing" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Program slicing, source coede analysis and manipulation, progrmadependence, dependence graph

05451 Executive Summary – Beyond Program Slicing

The aim of the "beyond program slicing" seminar was to explore emergent applications of program slicing and ways in which slicing techniques and ideas could be combined with those from other areas of program analysis and manipulation.

Keywords: Summary

Joint work of: Harman, Mark; Binkley, Dave; Krinke, Jens

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/584

05451 Group 4 Discussion – Popularizing Slicing

Christian Lindig (Universität Saarbrücken, D)

This report summarizes the results of the discussions of our working group at the Dagstuhl Seminar, Beyond Program Slicing.

The group aimed to answer the question "Why is slicing not well-known and only rarely used outside the slicing research community?".

Current implementations of slicing algorithms are language specific and somewhat monolithic: they are hard to extend and hard to disentangle from supporting infrastructure. This limits the service of these implementations both to the slicing research community and a wider audience of programmers in general. To address these factors we proposed two ideas. Our first proposal is to implement a general purpose slicing tool in the Datalog language. The algorithm for slicing could be succinctly and intuitively described in Datalog: the language optimizations giving an efficient implementation. Other often-used techniques, such as points-to analysis, have been or would easily be specified in Datalog. This would engender a modularized and flexible environment with good separation of the algorithmic aspects from the infrastructure. This would allow various modules to be used together and various tweaks to the algorithms to be readily implemented and explored. These properties make a Datalog based slicing tool a promising choice for both research and education. The other idea for popularizing slicing is to include backward (dynamic) slicing in the implementation of runtime environments of suitable programming languages. This way slices would be made accessible from within programs - just as stack traces already are in the Java and .NET platforms. At least, such a solution can help to improve error messages and debugging. But we expect more. We hope that slicing-improved systems would familiarize programmers with slicing techniques. This would then allow programmers to give feedback to the research community on further applications and extensions of slicing.

In summary, the Datalog tool addresses the needs of the research community. It provides a nice environment to get directly to the intellectual core of slicing; allowing for development of algorithms and integration with related analyses. Integration of slicing technology into the runtime environments of programming languages brings slicing into the programmer's hands, independently of their choice of development tools.

Joint work of: Lindig, Christian; Howroyd, John; Kiss, Akos

05451 Group 5 - Bananas, Dark Worlds, and AspectH

Silvia Breu (Universität Saarbrücken, D)

This report summarises our idea of code clone detection in Haskell code and refactorings based on identified clones as it evolved in our working group-of-three discussion at the Dagstuhl seminar "Beyond Program Slicing".

Keywords: Haskell, code clone detection, refactoring, functional control graph

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/491

Using Program Slicing to Identify Faults in Software

Sue Black (London South Bank Univ. - London, GB)

This study explores the relationship between program slices and faults.

The aim is to investigate whether the characteristics of program slices can be used to identify fault-prone software components. Slicing metrics and dependence clusters are used to characterise the slicing profile of a software component, then the relationship between the slicing profile of the component and the faults in that component are then analysed. Faults can increase the likelihood of a system becoming unstable causing problems for the development and evolution of the system. Identifying fault-prone components is difficult and reliable predictors of fault-proneness not easily identifiable. Program slicing is an established software engineering technique for the detection and correction of specific faults. An investigation is carried out into whether the use of program slicing can be extended as a reliable tool to predict fault-prone software components. Preliminary results are promising suggesting that slicing may offer valuable insights into fault-proneness.

Keywords: Program slicing, slicing metrics, fault proneness, software quality

Joint work of: Black, Sue; Counsell, Steve; Hall, Tracy; Wernick, Paul

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/587

And now for something completely different...

Sue Black (London South Bank Univ. - London, GB)

A pilot experiment was conducted at Dagstuhl using the 'Beyond program slicing' seminar attendees. Attendees were split into three groups: all were given the same program to understand and a list of program comprehension related questions to answer. Group one had only the source code, group two had the source code and the dynamic trace of the program, group three had the source and a control-flow graph of the program.

Keywords: Group experiment, program comprehension, source code, dynamic trace, control flow graph

Joint work of: Black, Sue; Bouillon, Philipp; Ducasse, Mireille

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/583

Extanding C Global Surveyor

Silvia Breu (Universität Saarbrücken, D)

Software failure are noted for their blowing large sums of money and sometimes even human life, in particular in the area of safety critical mission software. The most well-known desaster happened in 1996 when Ariane 501 exploded shortly after launch. The least it did was to cost the European space program half a billion US$ due to an overflow in an arithmetic conversion.

The Automated Software Engineering Group at the NASA Ames Research Center has developed C Global Surveyor (CGS), a static analysis tool based on abstract interpretation. It particularly concentrates on runtime errors that are hard to find during development such as out-of-bound array accesses, acesses to non-initialised variables, and de-references of null pointers. CGS proved to analyse large, pointer intensive and heavily multithreaded code (up to 280 KLoC) in a couple of hours with a constant precision of 80%. It is used to successfully analyse mission-critical flight software of NASA's "Mars Path-Finder" (MPF) and Deep Space 1 (DS1) legacy as well as software of the Mars Exploration Rover (MER) mission (650 KLoC) and other JPL-based missions.

However, the abstract interpretation techniques on which CGS is based, need to be augmented by complimentary program analysis techniques in order to enhance CGS and support the developer when analysing very large systems. As a first step, we included the construction of control flow graphs that represent the programs to be analysed. It is a first step towards the application of more advanced techniques such as program slicing.

Keywords: Static program analysis, abstract interpretation, program slicing

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/487

Domain-specific Slices

Magiel Bruntink (CWI - Amsterdam, NL)

Programs in dire need of some form of slicing are those whose concerns are implemented predominantly by idioms. Programmers working on such programs want to be able to obtain all the code related to specific concerns, in order to improve their understanding of the program, or to apply transformations to it. Code for concerns such as exception handling, when implemented in C by idiom, are not accurately obtained using traditional slicing methods, i.e., closures of control/data flow relations. Exception handling code is often tightly interconnected, or tangled, with other code, and therefore traditional slices tend to contain both exception handling and other code. The distinction between exception handling code and other code must therefore be made at a higher level of semantical abstraction.

We have done some work in the area of bug finding in an idiomatically implemented exception handling mechanism. Our verification tool, implemented as a state machine that traverses all program paths, is capable of accurately emitting exception handling code as a side effect. The result is a collection of program points that are traversed by the tool within a certain state, for instance, all the code that is executed after a certain error has occurred. We intend to use such "domain-specific" slices to improve the programmers' code understanding, or to facilitate automatic transformations. In my talk I will give a short overview of this work, and highlight my ideas about the applications of domain-specific slices.

Keywords: Slicing idioms

Association rules for fault localization

Peggy Cellier (IRISA - Rennes, F)

The current trend in debugging and testing is to cross-check information collected during several executions. Jones et al., for example, propose to use the instruction coverage of passing and failing runs in order to visualize suspicious statements [Jones, Harrold, Stasko - icse2002].

It has been shown that the method of Jones et al. can be re-interpreted as a data mining procedure. More particularly, they define a metric which characterizes association rules between the execution of a statement and failure (stmti -> Fail) [Denmat,Ducasse,Ridoux - ASE 2005].

Association rules produce some sort of non-executable slices. They are smaller than executable slices and they have a semantics, namely "when statements i, j, ... k are executed the execution fails most of the time." Another advantage is that the analysis is not dependent on the semantics of the languages in which the program is written, only the choice of attributes for the analysis is.

We are currently investigating the use of association rules in their general form, namely with several attributes on each side of the rules.

Keywords: Debugging, data mining, association rules

Joint work of: Cellier, Peggy; Ducassé, Mireille; Ridoux, Olivier; Ferré, Sébastien

Conditioned Slicing

Sebastian Danicic (Goldsmiths College - London, GB)

One aim of slicing is to simplify programs so they are easier to understand.

One problem is that conventional slices, becauseof their size arenot much easier to understand than the original.

In conditioned slicing we further simplify our programs by scattering them with assertions at different program points. The conditoned slicer, using a combination of symbolic execution and theorThis has the potential to make slices much smaller and hence easier to understand. Future work will add inductive reasoning with loop invariants to improve conditioning.

Sequential Recomposition Slicing

Ran Ettinger (Oxford University, GB)

Extracting a slice is easy. Extracting a slice, without duplicating it, is not as easy. The result is a sequential composition of the slice and its "complement" (borrowing Gallagher's terminology). Here, the main question is: "what is the complement?"

According to Lakhotia and Deprez, it is a union of backward slices (from all non-extracted substatements). This has been criticised (by Komondoor and Horwitz) for causing too much code duplication, due to lack of dataflow (from the slice to its complement). I propose to remedy this limitation by introducing yet another kind of slicing. I (tentatively) call it "sequential-recomposition slicing", and suggest accordingly an improved slice extraction transformation, called "sequential-recomposition".

The sequential-recomposition has been developed using a formal framework for proving correctness of slicing-based code motion refactorings. This opens the way for a systematic study of a family of refactorings and gives confidence in the correctness of this highly non-trivial transformation.

Keywords: Slicing, slice-extraction, refactoring, code-motion, sequential-recomposition

Beyond Slicing: Program Sliding

Ran Ettinger (Oxford University, GB)

Extracting a slice without duplicating it, in an attempt to improve separation of concerns in existing code, is not trivial. The result is a sequential composition of the slice and its 'complement' (borrowing Gallagher's terminology). Here, the main question is: "what is the complement?" According to Lakhotia and Deprez, it is a union of backward slices (from all non-extracted substatements). This has been criticised (by Komondoor and Horwitz) for causing too much code duplication, due to lack of dataflow (from the slice to its complement). I propose to remedy this limitation by introducing yet another kind of slice: a 'complement-slice', or 'co-slice'. The case for co-slice is given by example, using a novel program representation of transparency slides. With slides, the separation of a slice from its co-slice is illustrated as an operation of program sliding.

Keywords: Slicing, sliding, sequential recomposition, co-slice

And-Or Dependence Graphs for Slicing Statecharts

Chris Fox (University of Essex, GB)

The construction of an And-Or dependence graphs is illustrated, and its use in slicing statecharts is described. The additional structure allows for more precise slices to be constructed in the event of additional information, such as may be provided by static analysis and model checking, and with constraints on the global state and external events.

Keywords: Slicing, statecharts, And-Or depdence graphs, interference, conditioning

Joint work of: Fox, Chris; Luangsodsai, Arthorn

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/493

"Real" Decomposition Slicing

Keith B. Gallagher (Loyola College - Baltimore, USA)

This position paper is an attempt to use well-understood mathematical concepts to attack the problem of code evolution. Using linear algebra, we can decompose a vector space into a direct-sum decomposition of invariant subspaces. That is, a transformation $FX = B$ can be represented, with a suitable basis transformation, as $F = F_1 \bigoplus F_2 \bigoplus \cdots F_n$ such that $F_i X \subseteq F_i$. *If* we could apply this method to program transformation, we then have two new ways to to comprehend the evolution problem. First, a change to F could be written as $F_1 \bigoplus F_2 \bigoplus \cdots F_i' \cdots \bigoplus \cdots F_n$, where F_i' is the change. Second, an extension (enhancement) to A could be written as $F_1 \bigoplus F_2 \bigoplus \cdots F_n \bigoplus E_1$, where E_1 is the enhancement. The question is: Is slicing up to this?

Keywords: Program slicing, decomposition slicing, direct sum decomposition

Precise Slicing of Java Programs

Christian Hammer (Universität Passau, D)

A new, improved algorithm for Java is presented. The object-sensitive approach for slicing object-oriented languages did not present a coherent algorithm for dealing with (nested) objects as parameters. The new algorithm presented here always generates correct and precise slices for object parameters. In particular, it contains a criterion for the safe termination of unfolding recursive data structures. Furthermore a new approach for May-Happen-In-Parallel analysis is presented. It is formulated in a high-level language "datalog" and extends the original MHP analysis to Java semantics including recursion, dynamic thread

creation and must-alias analysis. To do so, it is based on a very precise context-sensitive Points-To analysis.

The presented analysis could reduce the number of interference dependence edges drastically, allowing more precise slices of threaded Java programs.

Keywords: Static Program Slicing, Java, Object Trees

Full Paper:
http://www.infosun.fmi.uni-passau.de/st/papers/paste04/

See also: The first part of this work was published at PASTE 2004

A New Foundation for Control Dependence and Slicing

John Hatcliff (Kansas State University, USA)

The notion of control dependence underlies many program analysis and transformation techniques. Despite wide application, existing definitions and approaches to calculating control dependence are difficult to apply directly to modern program structures because modern applications make substantial use of exception processing and increasingly support reactive systems designed to run indefinitely.

This talk revisits foundational issues surrounding control dependence and develops definitions and algorithms for computing several variations of control dependence that can be directly applied to modern program structures. To provide a foundation for slicing reactive systems, the paper proposes a notion of slicing correctness based on weak bisimulation and proves that the new definitions of control dependence generate slices that conform to this notion of correctness.

This new framework of control dependence definitions and correctness results is able to support programs with either reducible or irreducible control flow graphs. Finally, a variety of properties show that the new definitions conservatively extend classic definitions. These new definitions and algorithms form the basis of Indus Java Slicer – a publicly available program slicer that has been implemented for full Java.

Joint work of: Hatcliff, John; Ranganath, Venkatesh Prasad; Amtoft, Torben; Banerjee, Anindya; Dwyer, Matthew B.

Full Paper:
http://indus.projects.cis.ksu.edu

See also: This work was published at ESOP 2005

Tool Demo: The Indus Program Slicer and Kaveri Eclipse-based User Interface

John Hatcliff (Kansas State University, USA)

This demo presents a modular program slicer for full Java called Indus Java program slicer and a Eclipse-based user interface for Indus called Kaveri.

Indus is a library of classes that enables users to quickly assemble a highly customized non-SDG based inter-procedural program slicer capable of slicing concurrent Java programs. The core library contains core slicing engine along with post processing transformations such as residualization and executability injection to enable program slicing for purposes with diverse requirements such as program comprehension and model checking. The accompanying libraries from Indus provide well-defined interfaces (along with implementations) to various static analyses such as points-to analysis, various dependence analyses, escape analysis, monitor analysis, etc, that are required by program slicing.

Kaveri is an eclipse plugin that relies on the above library to deliver program slicing to eclipse platform. Apart from the basic feature for generating program slices from within eclipse along with an intuitive UI to view the slice, the plugin also provides the capability for chasing various dependences in the application to understand the slice.

Joint work of: Ranganath, Venkatesh; Jayaraman, Ganeshan; Hatcliff, John

Full Paper:
http://indus.projects.cis.ksu.edu

Advanced Stratego with some thoughts on the connection with slicing

Karl Trygve Kalleberg (University of Bergen, N)

I want to present some of the advanced language features of the Stratego/XT program transformation system, in particular concrete syntax, dynamic rules and extensible language definitions, and go on to highlight the areas we have applied this system to.

Particularly relevant to our context is the framework we have built for generic dataflow analysis and dead code elimination, which we already use for some forms of slicing. I also want to present the work done in the Stratego group on the Dryad open compiler framework for Java, and discuss how this may be useful for program slicing of Java code, also outside the Stratego group.

Some questions that I want to discuss in this context are: is there anything we should add/change in Stratego or the XT environment so that implementing slicers becomes easier; what should we keep in mind on slicing when designing our Java open compiler?

Keywords: Stratego, Program Transformation

Making Slicing Mainstream: How can we be Weiser?

Karl Trygve Kalleberg (University of Bergen, N)

By now, the concept of program slicing has been known in the research community for around 25 years. As a research topic, it has enjoyed a fair share of popularity, evidenced by the number of articles published on the topic following Mark Weiser's seminal paper. However, outside research circles, program slicing appears to be virtually unknown.

In this report, we take the premise that program slicing is both technically relevant, and has a sufficient theoretical foundation, to be applied in practice within the software industry. With this premise in mind, we ask ourselves, "what are the mechanisms by which we as a community could make program slicing mainstream"?

Keywords: Program Slicing, Popularization

Joint work of: Kalleberg, Karl Trygve; Hall, Tracy; Ettinger, Ran

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/486

Formalizing Slicing - Results, Current Work and Challenges

Ákos Kiss (University of Szeged, H)

Based on "common knowledge", people often expect that a static slice is a valid dynamic slice as well, even if overly large. However, there are counter examples which show that it's not always true. The reason for this is in the definitions of the slicing techniques (Weiser's static slicing does not care about the path of execution, while the dynamic slicing as defined by Korel and Laski does). The problem is that our "common knowledge" does not fit to the original definitions. However, if we have no formal background how can we compare existing slicing techniques, or how can we reason about them?

We have chosen the program projection theory, which is based on syntactic orderings and semantic equivalence relations on programs, as the basis of our work. This work aims at building up a unified formal theory of slicing, where different slicing techniques can be compared. So far, we have been able to formalize the two most important slicing techniques, i.e., the static and the KL-dynamic one, and some new ones as a "side effect". With the help of this formalization, we have been able to compare these techniques in the subsumes relation (which describes whether slices of one type are always valid slices of another type of slicing) and we have been able to rank the techniques, based on the size of minimal slices they make possible.

Now, that the basis of a unified theory has been set, we are facing new challenges. We have to incorporate new techniques and concepts, like amorphous slicing, dependence-based slicing, conditioned slicing, forward slicing, nonterminating programs and transfinite loops. To address the emerged issues, we set out a manifeto, a research agenda, which we are willing to follow in order to reach our goals.

Keywords: Slicing, formalization, unified theory

Joint work of: Kiss, Ákos; Binkley, Dave; Danicic, Sebastian; Gyimóthy, Tibor; Harman, Mark; Korel, Bogdan

Information Flow Control for Java Based on Path Conditions in Dependence Graphs

Jens Krinke (FernUniversität in Hagen, D)

Language-based information flow control (IFC) is a powerful tool to discover security leaks in software. Most current IFC approaches are however based on non-standard type systems. Type-based IFC is elegant, but not precise and can lead to false alarms.

We present a more precise approach to IFC which exploits 15 years of research in static program analysis. Our IFC approach is based on path conditions in program dependence graphs (PDGs). PDGs are a sophisticated and powerful analysis device, and today can handle realistic programs in full C or Java. We first recapitulate a theorem connecting the classical notion of noninterference to PDGs.

We then introduce path conditions in Java PDGs. Path conditions are necessary conditions for information flow; today path conditions can be generated and solved for realistic programs. We show how path conditions can produce *witnesses* for security leaks.

The approach has been implemented for full Java and augmented with classical security level lattices. Examples and case studies demonstrate the feasibility and power of the method.

Joint work of: Hammer, Christian; Krinke, Jens; Snelting, Gregor

Automated Compiler Testing

Christian Lindig (Universität Saarbrücken, D)

Code generated by a compiler must obey invariants and any violation of an invariant indicates a bug in the compiler. We can automate the test of compilers by generating source code randomly that tests such invariants when executed. Testing then becomes generating code, compiling, and running it without supervision. Once we find the violation of an invariant we also have an executable test

case that demonstrates the compiler bug. However, such a test case is typically hard to understand because it is machine generated and contains superfluous details. By tying the test code generator and the compiler into a feedback loop we can further minimize such a test case: the code generator removes details from the test case as long as it is still violating an invariant. This is akin to dynamic slicing that also identifies the parts of a program responsible for a given effect.

The final result is a method to automatically obtain bug reports that are both executable and minimal.

Keywords: Compiler, random testing, calling convention, bugs, gcc, language C, invariants

Slicing criteria – moving towards higher levels of abstraction

Jürgen Rilling (Concordia University - Montreal, CDN)

Over the last 30+ years, since the first introduction of static slicing by Mark Weiser, a variety of new and improved slicing algorithm have been developed. Common for most of these algorithms is their focus on source code and slicing criteria that closely match the slicing criterion introduced by Weiser, a variable at some point of interest.

For program slicing to be accepted by a larger community within the software engineering, program comprehension domain community, it will be essential to derive new slicing criteria, at different abstraction levels to support new application areas. In this presentation a brief overview of some of the existing slicing algorithms and their slicing criterion will be given followed by a presentation of Use Case Map (UCM) slicing. UCM slicing supports as slicing criteria any artifacts of a UCM diagrams and therefore allows the slicing of software specification. The presentation concluded with a discussion on other levels of abstractions program slicing can be extended to as well as other types of slicing criteria.

Architectural Slicing

Nuno Miguel Rodrigues Feixa (University of Minho - Braga, P)

Slicing, understood as a family of techniques for identifying and isolating program fragments that depend on or are depended upon by a specific entity, plays a major role in program comprehension and re-engineering. Do such techniques scale from the micro, program-oriented level, to the macro, architectural one? Such is the purpose of my research programme, which encompasses the development of generic slicing algorithms, applicable to large, multi-language, heterogeneous systems as well as new techniques to extract and represent architectural patterns and decisions from legacy systems and to reason formally upon them.

Our research efforts so far targeted the development of specific implementations of slicing algorithms for functional programs, a somewhat neglected area in terms of program understanding. The results achieved include the development of a functional slicer for Haskell, involving suitable extraction algorithms and appropriate intermediate data representations, and its application in the context of a methodology for component identification and software clustering. Another research topic, still in a preliminary phase, concerns the development of slicing techniques by calculation, i.e., which resort to standard program calculation strategies, based on the so-called Bird-Meertens formalism. The slicing criterion is specified either as a projection or a hiding function which, once composed with the original program, leads to the identification of the intended slice.

Keywords: Software Architecture

Slicing Functional Programs by Calculation

Nuno Miguel Feixa Rodrigues (University of Minho - Braga, P)

Program slicing is a well known family of techniques used to identify code fragments which depend on or are depended upon specific program entities. They are particularly useful in the areas of reverse engineering, program understanding, testing and software maintenance.

Most slicing methods, usually targeting either the imperative or the object oriented paradigms, are based on some sort of graph structure representing program dependencies. Slicing techniques amount, therefore, to (sophisticated) graph transversal algorithms.

This paper proposes a completely different approach to the slicing problem for functional programs. Instead of extracting program information to build an underlying dependenciesŠ structure, we resort to standard program calculation strategies, based on the so-called Bird- Meertens formalism. The slicing criterion is specified either as a projection or a hiding function which, once composed with the original program, leads to the identification of the intended slice. Going through a number of examples, the paper suggests this approach may be an interesting, even if not completely general alternative to slicing functional programs.

Keywords: Program Slicing, Algebra of Programming, Functional Programming

Joint work of: Rodrigues, Nuno Miguel Feixa; Barbosa, Luís S.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/484

Generic Slicing on Machine Code

Marc Schlickling (Universität Saarbrücken, D)

The complexity of software used in embedded systems grows rapidly. Moreover, embedded software is often subject to strict safety constraints. Thus, there is an urgent need to ensure the safeness and correctness of such a software system. Due to compiler transformations and optimizations, guaranteeing these characteristics can only be achieved at machine code level.

Programs checking a desired behaviour are rare and are - as a result of the software's complexity - no one button tools. They require some help from the developers. Furthermore program understanding at machine code level is hard, so there is a need for supplementary tools helping the developers. In many cases, slicing supports finding relevant parts of a program and thus, understanding it.

Caused by the huge number of embedded processors with different instruction sets, there is a need for a generic slicing algorithm. This talk presents a solution for slicing executable programs also taking memory accesses into account.

On the Impact of Alias Analysis Precision on Slicing

Markus Schordan (TU Wien, A)

The applications of program slicing and improvemements to algorithm speed and precision have found a great interest in the slicing community. But only few publications address pointers as a major source of imprecision in slicing of real-world applications. One reason is that alias analysis can be treated as a separate problem where a slicing algorithm only uses the results of an alias analysis. Another reason is that combined approaches, where a slicing algorithm is extended to cope with all aspects of aliasing, becomes quite complicated which makes it difficult to communicate the form or level of imprecision of the implicit alias analysis to a user in a slicing tool; this scenario becomes worse the more precise the semantics of strongly typed pointers, unconstrained pointers, function pointers or in object-oriented languages virtual functions, are modeled with precise semantics. In particular it is a problem how precise slices can be visualized with its corresponding possible different heap configurations that actually impact the slice's size.

We present a combination of static alias analysis with static slicing. The alias analyses of interest are different forms of shape analyses that lend themselves for visualization and easier understanding of the different possible heap configurations in the execution of a program. The combined computation of shape information and other analysis information is presented, where we focus on the information impacting the size of a slice.

Keywords: Data-Flow Analysis, Executable Slice, Flow-Shape Graphs, Shape Analysis, Static Analysis Tools

Probabilistic Program Slicing - introduction

Jeremy Singer (Manchester University, GB)

I would like to discuss the idea of probabilistic program slicing.

Recently, there has been a growing interest in probabilistic program analysis of different kinds.

I hope to give an initial formulation for a probabilistic version of program slicing. I will highlight some example applications, including analysis support for effective speculative thread-level parallelism.

Probabilistic Program Slicing - slides for Monday talk

Jeremy Singer (Manchester University, GB)

Probabilistic Program slicing is a proposal for a new form of slicing. In this talk we will see that other program analyses have been adapted to handle probabilities. We will provide motivation for a probabilistic form of slicing. Then we give a simple walk-through example of probabilistic slicing. We conclude with a survey of the future work required.

Keywords: Probability, slicing

Towards Probabilistic Program Slicing

Jeremy Singer (Manchester University, GB)

This paper outlines the concept of *probabilistic* program slicing. It walks through a simple example before describing some algorithmic concerns. Then three motivating applications are described.

Finally it highlights existing work that may be built upon, and future work that needs immediate attention if this idea is to succeed.

Keywords: Probability, slicing, speculation

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/485

Concern Slicing

Tom Tourwé (CWI - Amsterdam, NL)

Magiel Bruntink and I are working on the same topic, so he will present something and I will probably only show a demo ...

Attribute Slicing

Neil Walkinshaw (Sheffield University, GB)

It can often be the case in object-oriented programming that classes bloat, particularly if the represent an ill-formed abstraction. A poorly formed class tends to be formed from disjoint sets of methods and attributes. This can result in a loss of cohesion within the class. Slicing attributes can be used to identify and make explicit the relationships between attributes and the methods that refer to them. This can be a useful tool for identifying code smells and ultimately refactoring. Attribute slicing can also be used to examine the relationships between attributes, which in turn could be useful for reverse engineering object state machines.

Keywords: Slicing, object-oriented programming, refactoring

Using Attribute Slicing to Refactor Large Classes

Neil Walkinshaw (Sheffield University, GB)

It can often be the case in object-oriented programming that classes bloat, particularly if the represent an ill-formed abstraction. A poorly formed class tends to be formed from disjoint sets of methods and attributes. This can result in a loss of cohesion within the class. Slicing attributes can be used to identify and make explicit the relationships between attributes and the methods that refer to them. This can be a useful tool for identifying code smells and ultimately refactoring. Attribute slicing can also be used to examine the relationships between attributes, as is the case in decomposition slicing. This paper introduces attribute slicing in the context of refactoring bloated classes.

Keywords: Refactoring, cohesion, object-oriented slicing

Joint work of: Kirk, Douglas; Roper, Marc; Walkinshaw, Neil

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2006/490