Abstract : 1

SLR: Path-Sensitive Analysis through Infeasible-Path Detection and Syntactic Language Refinement

<http://pages.cs.wisc.edu/~bgogul/Research/Papers/sas08.slr.pdf>

We present a technique for detecting semantically infeasible paths in programs using abstract interpretation. Our technique uses a sequence of path-insensitive forward and backward runs of an abstract interpreter to infer paths in the control ﬂow graph that cannot be exercised in concrete executions of the program. We then present a syntactic language reﬁnement (SLR) technique that automatically excludes semantically infeasible paths from a program during static analysis. SLR allows us to iteratively prove more properties. Speciﬁcally, our technique simulates the eﬀect of a path-sensitive analysis by performing syntactic language reﬁnement over an underlying path insensitive static analyzer. Finally, we present experimental results to quantify the impact of our technique on an abstract interpreter for C programs.

Abstract : 2

Heuristics-based infeasible path detection for dynamic test data generation

Automated test data generation plays an important part in reducing the cost and increasing the reliability of software testing. However, a challenging problem in path-oriented test data generation is the existence of infeasible program paths, where considerable effort may be wasted in trying to generate input data to traverse the paths. In this paper, we propose a heuristics-based approach to infeasible path detection for dynamic test data generation. Our approach is based on the observation that many infeasible program paths exhibit some common properties. Through realizing these properties in execution traces collected during the test data generation process, infeasible paths can be detected early with high accuracy. Our experiments show that the proposed approach efficiently detects most of the infeasible paths with an average precision of 96.02% and a recall of 100% of all the cases.

Abstract : 3

Automatic detection of infeasible paths in software testing

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5585610"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5585610"arnumber=5585610](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5585610)

A challenging problem in path-oriented test data generation is the presence of infeasible paths. Timely detecting these infeasible paths cannot only save test resources but also improve test efﬁciency. A popular method of detecting infeasible paths is to determine branch correlations, which is a difﬁcult task and usually cannot be done timely and exactly. In this study, the authors propose a method of automatically determining the branch correlations of different conditional statements, therefore detecting infeasible paths. First, some theorems are given to determine branch correlations based on the probabilities of the conditional distribution corresponding to different branches’ outcome (i.e. true or false); then, the maximum likelihood estimation is employed to obtain the values of these probabilities; ﬁnally, infeasible paths are detected according to branch correlations. The authors apply the proposed method in some typical programs, and the results show that the proposed method can accurately detect infeasible paths. The achievement provides an effective and automatic method of detecting infeasible paths, which is signiﬁcant in improving the efﬁciency of software testing.

Abstract : 4

Detecting large number of infeasible paths through recognizing their patterns

[http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"&HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"id=1287655HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"&HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"acc=ACTIVE%20SERVICEHYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"&HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAAHYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"&HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"CFID=240240844HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"&HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"CFTOKEN=99313165HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"&HYPERLINK "http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473"\_\_acm\_\_=1377034176\_e6dbe7190cbc57d3efd92039ced25473](http://delivery.acm.org/10.1145/1290000/1287655/p215-ngo.pdf?ip=206.180.215.228&id=1287655&acc=ACTIVE%20SERVICE&key=C2716FEBFA981EF104BAE1A30F6EE2281FAE65500E195DAA&CFID=240240844&CFTOKEN=99313165&__acm__=1377034176_e6dbe7190cbc57d3efd92039ced25473)

A great majority of program paths are found to be infeasible, which in turn make static analysis overly conservative. As static analysis plays a central part in many software engineering activities, knowledge about infeasible program paths can be used to greatly improve the performance of these activities especially structural testing and coverage analysis. In this paper, we present an empirical approach to the problem of infeasible path detection. We have discovered that many infeasible paths exhibit some common properties which are caused by four code patterns including *identical/complement-decision, mutually-exclusive-decision, check-then-do* and *looping-by-flag* pattern. Through realizing these properties from source code, many infeasible paths can be precisely detected. Binomial tests have been conducted which give strong statistical evidences to support the validity of the empirical properties. Our experimental results show that even with some limitations in the current prototype tool, the proposed approach accurately detects 82.3% of all the infeasible paths.

Abstract : 5

Using Branch Correlation to Identify Infeasible Paths for Anomaly Detection

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=4041840"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=4041840"arnumber=4041840](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=4041840)

In this paper, we propose a system called Infeasible Path Detection System (IPDS) to combat memory tampering attacks causing invalid program control flows. In our system, the compiler analyzes correlations between branches and then the analyzed information is conveyed to the runtime system. The runtime system detects dynamic infeasible program paths by combining compiler determined information with runtime information to check the legality of the path taken during execution. IPDS achieves a zero false positive rate and can detect a high percentage of memory tampering for many attacks in which the tampering actually causes a change in control flow. Moreover, IPDS only incurs a modest amount of hardware resource and negligible performance penalty.

Abstract : 6

Efficient Detection and Exploitation of Infeasible Paths for Software Timing Analysis

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=1688820"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=1688820"arnumber=1688820](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=1688820)

Accurate estimation of the worst-case execution time (WCET) of a program is important for real-time embedded software. Static WCET estimation involves program path analysis architectural modeling. Path analysis is complex due to the inherent difficulty in detecting and exploiting infeasible paths in a program's control flow graph. In this paper, we propose an efficient method to exploit infeasible path information for WCET estimation without resorting to exhaustive path enumeration. We demonstrate the efficiency of our approach for some real-life control-intensive applications.

Abstract : 7

Detection of infeasible paths using Presburger arithmetic

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=544614"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=544614"arnumber=544614](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=544614)

Detecting infeasible paths (IFPs) allows accurate computation of various kinds of program slices, and accurate detection of semantic errors that may occur when two variants of a program are merged. We propose a method of efficiently determining the truth of a prenex normal form Presburger sentence (P-sentence) bounded only by existential quantifiers, which is suitable for detecting IFPs. In this method, a coefficients matrix is converted into a triangular matrix based on the method proposed by Cooper (1972). If the rank of the matrix is lower than the degree of the matrix, the matrix is triangulated by using a method for solving one linear equation with three or more unknowns, so that the matrix can be back-substituted. This paper shows that an implementation of our method provides a slower increase in computation time than the previous method and reduces computation time by up to 3,000,000 times.

Abstract : 8

A Systematic Classification and Detection of Infeasible Paths for Accurate WCET Analysis of Esterel Programs

[http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.5523HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.5523&rep=rep1&type=pdf"&HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.5523&rep=rep1&type=pdf"rep=rep1HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.5523&rep=rep1&type=pdf"&HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.5523&rep=rep1&type=pdf"type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.5523&rep=rep1&type=pdf)

Synchronous programming languages like Esterel are widely used in safetycritical domains like avionics. However, it is only with the recent development of mature worst-case execution time (WCET) analysis tools that progress is being made on systematically studying the WCET analysis problem for languages like Esterel. In this context, we present techniques for methodically classifying and detecting different types of infeasible paths that arise while compiling Esterel programs into executable code, via high-level languages such as C. Our experimental results with well-known benchmarks show that the infeasible paths detected using our techniques result in as much as 36.5% reduction in the WCET estimates, compared to when no infeasible path detection is employed.

Abstract : 9

**A Symbolic Execution Tool Based on the Elimination of Infeasible Paths**

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110"arnumber=5615110](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110)

Software testing forms a substantial activity of the software development cycle. Although important, it lacks from being automated mainly because of the various undecidable problems that it encounters. To this extend efficient heuristics have been proposed in order to bypass this problem. One such approach, called symbolic execution, is usually used for automating the test data generation activity. In this paper, an automated symbolic execution tool is proposed. The tool employs an efficient path heuristic, integrated with random testing for producing test cases. The tool handles the path explosion and constraint solving problems efficiently. This is achieved by targeting on specific likely to be feasible paths and by using a linear programming approach for the determination of their feasibility. Preliminary results are very encouraging as they show that a high coverage can be achieved within a limited amount of time-effort.

Abstract : 10

Safe and Efﬁcient Elimination of Infeasible Execution Paths in WCET Estimation

[http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9083HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9083&rep=rep1&type=pdf"&HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9083&rep=rep1&type=pdf"rep=rep1HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9083&rep=rep1&type=pdf"&HYPERLINK "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9083&rep=rep1&type=pdf"type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9083&rep=rep1&type=pdf)

Reasoning about the timing properties of a program is indispensable in the development of time critical systems where failure to meet deadlines can result in loss of life or material. To this end having tools to calculate safe and tight Worst Case Execution Time (WCET) bounds can be very valuable. In most of the approaches to date a lot of pessimism is attributed to the fact that many paths that are infeasible are not excluded from the WCET computations. To remedy this, user annotations to the source code were proposed and used. Unfortunately, there is no guarantee that these annotations are always correct. This fact renders such a manual approach unacceptable in the case of R/T systems where safety is an absolute priority. In this paper another approach for the safe elimination of infeasible execution paths is presented. This method is based on the R/T programming language SIGNAL and its internal Dynamic Graph representation.

Abstract : 11

Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis using Abstract Execution

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=4032336"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=4032336"arnumber=4032336](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=4032336)

Static Worst-Case Execution Time (WCET) analysis is a technique to derive upper bounds for the execution times of programs. Such bounds are crucial when designing and verifying real-time systems. A key component for statically deriving safe and tight WCET bounds is information on the possible program ﬂow through the program. Such ﬂow information can be provided manually by user annotations, or automatically by a ﬂow analysis. To make WCET analysis as simple and safe as possible, it should preferably be automatically derived, with no or very limited user interaction. In this paper we present a method for deriving such ﬂow information called abstract execution. This method can automatically calculate loop bounds, bounds for including nested loops, as well as many types of infeasible paths. Our evaluations show that it can calculate WCET estimates automatically, without any user annotations, for a range of benchmark programs, and that our techniques for nested loops and infeasible paths sometimes can give substantially better WCET estimates than using loop bounds analysis only.

Abstract : 12

Safe and Efficient Elimination of Infeasible Execution Paths in WCET Estimation

Reasoning about the timing properties of a program is indispensable in the development of time critical systems where failure to meet deadlines can result in loss of life or material. To this end having tools to calculate safe and tight Worst Case Execution Time (WCET) bounds can be very valuable. In most of the approaches to date a lot of pessimism is attributed to the fact that many paths that are infeasible are not excluded from the WCET computations. To remedy this, user annotations to the source code were proposed and used. Unfortunately, there is no guarantee that these annotations are always correct. This fact renders such a manual approach unacceptable in the case of R/T systems where safety is an absolute priority. In this paper another approach for the safe elimination of infeasible execution paths is presented. This method is based on the R/T programming language SIGNAL and its internal Dynamic Graph representation.

Abstract : 13

A Symbolic Execution Tool Based on the Elimination of Infeasible Paths

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110&tag=1"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110&tag=1"arnumber=5615110HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110&tag=1"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110&tag=1"tag=1](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=5615110&tag=1)

Software testing forms a substantial activity of the software development cycle. Although important, it lacks from being automated mainly because of the various undecidable problems that it encounters. To this extend efficient heuristics have been proposed in order to bypass this problem. One such approach, called symbolic execution, is usually used for automating the test data generation activity. In this paper, an automated symbolic execution tool is proposed. The tool employs an efficient path heuristic, integrated with random testing for producing test cases. The tool handles the path explosion and constraint solving problems efficiently. This is achieved by targeting on specific likely to be feasible paths and by using a linear programming approach for the determination of their feasibility. Preliminary results are very encouraging as they show that a high coverage can be achieved within a limited amount of time-effort.

Abstract : 14

Performance Validation Through Implicit Removal of Infeasible Paths of the Behavioral Description

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=6523665"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=6523665"arnumber=6523665](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=6523665)

In this paper we present a novel algorithm to identify infeasible paths in the behavioral code. The proposed approach initially partitions the behavioral code into segments. At each code segment it stores feasible paths implicitly. It also stores collections of input assignments which are derived using selected statements in the code segment. The method requires state-of-the-art data structures to store feasible paths and the required functions. Experimental results demonstrate the scalability of the proposed method.

Abstract : 15

Identification of Potentially Infeasible Program Paths by Monitoring the Search for Test Data

[http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=873665"&HYPERLINK "http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=873665"arnumber=873665](http://ieeexplore.ieee.org.ezproxy.gannon.edu/stamp/stamp.jsp?tp=&arnumber=873665)

A tool and techniques are presented for test data generation and identification of a path’s likely unfeasibility in structural software testing. The tool is based on the Dynamic Technique and search using Genetic Algorithms. Our work introduces a new fitness function that combines control and data flow dynamic information to improve the process of search for test data. The unfeasibility issue is addressed by monitoring the Genetic Algorithm’s search progress. An experiment shows the validity of the developed solutions and the benefit of using the tool.