Journal of Software Testing, Verification, and Reliability, V. 10, no. 3, September, 2000

An Empirical Investigation of the Relationship Between Spectra Differences and Regression Faults

Mary Jean Harrold,[†] Gregg Rothermel,[‡] Kent Sayre,[‡] Rui Wu,[*] Liu Yi[‡]

[†]College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
harrold@cc.gatech.edu

[‡]Department of Computer Science
Oregon State University
Corvallis, OR 97331
{grother,ksayre,liuyi}@cs.orst.edu

[*]Department of Computer and
Information Science
Ohio State University
Columbus, OH 43210
rwu@cis.ohio-state.edu

Abstract

Many software maintenance and testing tasks involve comparing the behaviors of program versions. Program spectra have recently been proposed as a heuristic for use in performing such comparisons. To assess the potential usefulness of spectra in this context an experiment was conducted, examining the relationship between differences in program spectra and the exposure of regression faults (faults existing in a modified version of a program that were not present prior to modifications, or not revealed in previous testing), and empirically comparing several types of spectra. The results reveal that certain types of spectra differences correlate with high frequency — at least in one direction — with the exposure of regression faults. That is, when regression faults are revealed by particular inputs, spectra differences are likely also to be revealed by those inputs, though the reverse is not true. The results also suggest that several types of spectra that appear, analytically, to offer greater precision in predicting the presence of regression faults than other, cheaper, spectra may provide no greater precision in practice. These results have ramifications for future research on, and for the practical uses of, program spectra.

Keywords: program spectra, software testing, empirical studies

1 Introduction

Various software testing and maintenance tasks require comparisons of the behaviors of program versions. For example, when a program is modified, regression testing is used to compare the behavior of the modified version to the behavior of its previous version, in the hope of detecting regression faults – faults existing in a modified version of a program that were not present prior to modifications, or not revealed in previous testing. Similarly, when a modified program fails, the behaviors of versions are compared in the hope of locating the cause of that failure. Tasks such as these constitute a significant percentage of the costs of software testing and maintenance, and thus, techniques that reduce the costs of these tasks are valuable.

Path spectra were recently proposed as a heuristic for assessing the magnitude of the behavioral changes between program versions [11].[1] A *path spectrum* is a distribution of paths derived from an execution of a program using program profiling. A *path-spectra-comparison technique* compares path spectra to gain an understanding of program behavior. Such a technique may aid in addressing testing and maintenance tasks that require such understanding. For example, constructing expected outputs for programs can be costly. If the presence of regression faults is reliably indicated by the presence of spectra differences, then during regression testing testers could identify the test cases on which a program and modified version exhibit spectra differences, and restrict the construction of expected outputs to those test cases. If the cost of constructing expected outputs for test inputs exceeds the cost of obtaining spectra for those inputs, this process may reduce regression testing costs. As a second example, spectra comparison may be a useful technique for locating points of divergence in computations, thus helping guide programmers in fault localization [11].

For path spectra to be useful in these contexts, however, they must provide meaningful behavior signatures. An assessment of the potential usefulness of path-spectra comparisons requires an understanding of the relationship between spectra differences and differences in program behavior.

Program behavior can be measured in many ways; however, one measure important to uses of spectra such as those just described involves whether particular inputs cause a program to fail. Reference [11] hypothesizes a strong association between spectra differences and failures, at least in one direction, stating that given a correct version of a program, and a faulty version of that program, one expects differences between spectra on inputs that cause the faulty version to fail. One goal of this work is to empirically investigate this claim.

If path spectra prove useful, then other spectra such as branch spectra or complete-path spectra may also be useful, and may provide a range of techniques, varying in cost and effectiveness, for examining program behavior. Reference [11] conjectures that spectra that track statement or branch executions will not be as useful as path spectra for distinguishing program behavior. There is some empirical data [1] to support this conjecture: this data indicates that path profiling data may be superior to branch profiling data for certain applications. On the other hand, another recent study [5] suggests the contrary. Neither of these studies, however, directly investigated program spectra. A second goal of this work is to perform such an investigation.

The next section of this paper defines program spectra more precisely, presents several different types of spectra, and compares these spectra types analytically. Section 3 describes the experiments, including their objectives, measures used, and experiment instrumentation and design. Section 4 presents the data collected in the experiments, and the analysis of that data. Section 5 presents additional discussion and conclusions.

[1] The primary use of spectra investigated in [11] involves the "Year 2000 problem," comparing spectra from *two runs of the same program* on input data that differs only with respect to date. The intuition is that spectra differences may help programmers locate date-dependent computations. The alternative use of spectra investigated here, in which spectra are collected from runs of a program and a modified version of the program on the same data, is suggested in [11] but not pursued in depth. The goal of this work is to empirically investigate this alternative suggestion.

Mnemonic	*Name*	*Description*
BHS	Branch Hit Spectra	conditional branches that were executed
BCS	Branch Count Spectra	number of times each conditional branch was executed
CPS	Complete Path Spectra	complete path that was executed
PHS	Path Hit Spectra	intraprocedural, loop-free path that was executed
PCS	Path Count Spectra	number of times each intraprocedural, loop-free path was executed
DHS	Data-dependence Hit Spectra	definition-use pairs that were executed
DCS	Data-dependence Count Spectra	number of times each definition-use pair was executed
OPS	Output Spectra	output that was produced
ETS	Execution Trace Spectra	execution trace that was produced

Table 1: A catalog of program spectra.

2 Program Spectra

A program spectrum characterizes, or provides a signature of, a program's behavior [11]. One class of program spectra, *path spectra*, uses path profiling [1, 4] to track the execution of loop-free intraprocedural paths in a program. Path spectra can track the frequency of path occurrences, or ignore frequency and track only whether or not paths were executed.

In addition to path spectra, many other signatures of program behavior are possible; for example, spectra can track the execution of statements or branches (outcomes of decisions). To obtain a broader empirical view of spectra, in this research, nine distinct types of spectra are considered (see Table 1). These spectra are next described, and examples are provided to illustrate them. In the following definitions, let P be a program.

Branch spectra. Branch spectra record the set of conditional branches that are exercised as P executes. The entry to a procedure is treated as a branch representing the "decision" to execute the procedure; this treatment enables the tracking of executions of procedures that contain no branches. If, for each conditional branch in P, the spectrum merely indicates whether or not that branch was exercised, the spectrum is a *branch-hit spectrum* (BHS). If, for each conditional branch in P, the spectrum indicates the number of times that branch was executed, the spectrum is a *branch-count spectrum* (BCS).

Complete-path spectrum. A *complete path spectrum* (CPS) records the complete path that is traversed as P executes. Such a spectrum may be intolerably expensive to utilize in practice; however, it provides a useful reference point for comparisons.

Path spectra. To provide some of the benefits of considering paths rather than statements or branches, while avoiding the expense of considering complete paths, Reps et al. [11] consider a different variety of spectra that tracks partial paths, which they refer to as "*path spectra*". Path spectra record the set of loop-free intraprocedural paths that are traversed as P executes. Path spectra were initially defined in [4], where an efficient algorithm for calculating and tracking loop-free intraprocedural paths is given.

The set of loop-free intraprocedural paths in a procedure is obtained by transforming the control flow graph[2] for that procedure into an acyclic control flow graph, as follows. Given a control flow graph G with

[2] A *control-flow graph* for program (or procedure) P is a directed graph G; nodes in G represent statements in P and edges in G represent the flow of control between those statements.

a unique entry node e and a unique exit node x, depth-first search is used to identify backedges (associated with cycles) in G. For each backedge (u, v), edges (e, u) and (v, x) are added to G. Finally, all backedges are removed from G. The resulting graph is acyclic, and contains a finite number of bounded-length paths. These are the paths utilized in path spectra.

If, for each such (loop-free, intraprocedural) path in G, a spectrum merely indicates whether or not that path was executed, the spectrum is a *path-hit spectrum* (PHS). If, for each such path in G, the spectrum indicates the number of times that path was executed, the spectrum is a *path-count spectrum* (PCS).

Data-dependence spectra. Data-dependence spectra record the set of definition-use pairs that are exercised as P executes.[3] If, for each definition-use pair in P, the spectrum merely indicates whether or not that definition-use pair was exercised, the spectrum is a *data-dependence-hit spectrum* (DHS). If, for each definition-use pair in P, the spectrum indicates the number of times that definition-use pair was exercised, the spectrum is a *data-dependence-count spectrum* (DCS).

Output spectrum. An *output spectrum* (OPS) records the output produced by P as it executes.

Execution-trace spectrum. An *execution-trace spectrum* (ETS) records the sequence of program statements traversed as P executes.

At first glance, CPS and ETS may appear identical, because they each involve complete control-flow paths through P. They differ, however, in that CPS does not record the actual instructions executed along that path, whereas ETS does.

OPS and ETS are of interest in this context because of their relationship to regression testing. One important regression testing activity is the selection of a subset of the test suite that was originally used to test a program, for use in testing a modified version of that program; Reference [13] provides details. In brief, given a program P, a test suite T for P, and a modified program version P', testers want to identify the test cases in T that reveal regression faults in P'. For test cases whose specified behavior has not changed, these "fault-revealing" test cases are exactly the test cases that produce different output spectra for P and P'. In general, there is no algorithm to precisely identify these test cases, but under certain conditions, the test cases that produce different execution-trace spectra constitute a conservative (safe) approximation. Several regression test selection techniques (e.g., [3, 6, 12]) exploit this relationship to select safe subsets of T for use in regression testing P'.

The motivation for considering DCS and DHS is that, intuitively, it seems likely that differences in executing definition-use associations might more closely correlate with differences in program behavior than differences in executing simple paths, statements, or branches.

To illustrate these spectra, program `Sums` and its control-flow graph are presented (Figure 1), with the spectra for `Sums` on two executions (Table 2): execution 1 uses input 10 and has expected output 0, and

[3] A *definition-use pair* is a tuple of the form (d, U, v) in which d is a statement in P and v is a variable, d (the definition) sets a value of v, U (the use) is either a non-predicate statement u that references v, or a pair (u, u') consisting of a predicate statement u that references v and a statement u' reached following evaluation of u, and there is a path in P from d to u without a redefinition of v.

```
program Sums
1 read i
2 sum = 0
3 while i < 10
4   read j
5   sum = sum + j
6   i = i + 1
  endwhile
7 print sum
end Sums
```

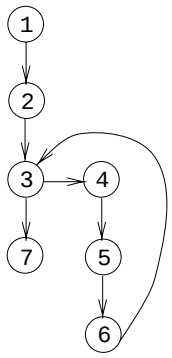

Figure 1: `Sums` and its control-flow graph.

Spectrum	Profiled entities	Execution 1 (input: 10)		Execution 2 (input: 8, 2, 4)	
		Hit	Count	Hit	Count
Branch	(1,2)	Y	1	Y	1
	(3,4)	N	0	Y	2
	(3,7)	Y	1	Y	1
Complete path	(1,2,3,7)	Y	NA	N	NA
	(1,2,3,(4,5,6,3)^2,7)	N	NA	Y	NA
Path	(1,2,3,7)	Y	1	N	0
	(1,3,7), (1,2,3,4,5,6,7), (1,3,4,5,6,7)	N	0	Y	1
Data-dependence	(1,(3,7),`i`), (2,7,`sum`)	Y	1	N	0
	(1,(3,4),`i`), (1,6,`i`), (6,(3,7),`i`), (2,5,`sum`), (5,7,`sum`),(6,6,`i`), (6,(3,4),`i`), (5,5,`sum`)	N	0	Y	1
	(4,5,`j`)	N	0	Y	2
Output	`sum` is 0	Y	NA	N	NA
	`sum` is 6	N	NA	Y	NA
Execution trace	(`read i,sum=0,while i<10,print sum`)	Y	NA	N	NA
	(`read i,sum=0,while i<10,(read j,sum=sum+j,i=i+1,while i<10)`^{2}`,print sum`)	N	NA	Y	NA

Table 2: Spectra for program `Sums` of Figure 1.

execution 2 uses inputs 8, 2, and 4 and has expected output 6. Where applicable, the two types of spectra in each category — hit and count — are shown in columns on the right in the table; where the spectra category does not have these subtypes, "NA" is listed. Consider, for example, the branch spectra for `Sums`. There are three conditional branches in `Sums` (recall that we consider entry edges to be branches), and the two executions exercise all of them: the BHS for execution 1 records Y for edges (1,2) and (3,7); the BHS for execution 2 records Y for edges (1,2), (3,4), and (3,7); the BCS for execution 1 records that edges (1,2) and (3,7) were each exercised once; and the BCS for execution 2 records that edges (1,2) and (3,7) were exercised once, and edge (3,4) was exercised twice.

Types of spectra can be analytically compared given the subsumption relationship that exists between them. Spectra type S_1 *subsumes* spectra type S_2 if and only if, whenever the S_2 spectra for program P, version P', and input i differ, the S_1 spectra for P, P', and i differ. Spectra type S_1 *strictly subsumes* spectra type S_2, denoted $\rightarrow$, if S_1 subsumes S_2, and for some program P, version P', and i, the S_1 spectrum differs but the S_2 spectrum does not. Spectra types S_1 and S_2 are *incomparable* if neither $S_1 \rightarrow S_2$ nor $S_2 \rightarrow S_1$. The following theorem establishes the subsumption relationship for all of the spectra under consideration.

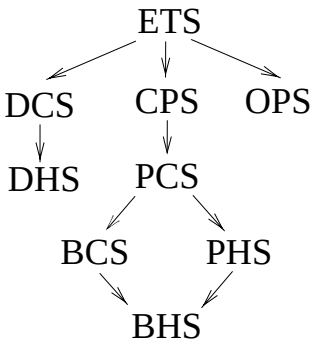

Figure 2: Spectra subsumption hierarchy.

Theorem 1. The family of spectra listed in Table 1 is partially ordered by strict subsumption as shown in Figure 2. Furthermore, spectra type S_1 strictly subsumes spectra type S_2 if and only if it is explicitly shown to do so in Figure 2 or follows from the transitivity of the relationship.

Proof: See Appendix A.

The spectra subsumption hierarchy shown in Figure 2 depicts an analytical relationship between spectra types that may help in assessing potential cost-benefit tradeoffs between these types. For example, the hierarchy suggests that PCS is more sensitive to differences in program behavior than BCS, and might be expected to more precisely reveal such differences. It is not clear, however, the extent to which this analytical expectation of greater effectiveness will be borne out in practice. The experiments described in this paper are intended, in part, to shed some light on this.

3 The Experiments

Two experiments were conducted: the first with a set of seven small (100 - 500 lines-of-code) C programs, and the second with a program an order of magnitude larger. The following sections describe the objectives, measures, instrumentation, and design shared in common by the two experiments. Subsequent sections in turn describe the results of the two experiments.

3.1 Objectives

The objective of these experiments was to investigate the following research questions:

1. Given program P, faulty version P' of P, and the universe of inputs U for P, what relationship exists between inputs that cause P and P' to produce different spectra and inputs that cause P and P' to exhibit different failure behavior? More precisely:

   (a) How often does an input $i \in U$ that exposes a regression fault in P' produce different spectra for P and P'?

(b) How often does an input $i \in U$ that produces different spectra for P and P' expose a regression fault in P'?

2. What are the relationships between the various spectra types, both in terms of their associations with program-failure behavior, and in terms of their associations with one another?

3.2 Measures

To quantify 1(a) and 1(b), two measures were utilized. Given program P, faulty version P', and the universe of inputs U for P, let $FR(P, P', U)$ be the set of inputs in U that expose a regression fault in P' (i.e., $FR(P, P', U)$ are Fault Revealing for P, P', and U). For each spectra variety S, let $SR(P, P', U)$ be the set of inputs in U that produce spectra on P and P' that differ (i.e., $SR(P, P', U)$ are spectrum S Revealing for P, P', and U).

Degree of Imprecision. For spectra type S, any input i in U that is in $SR(P, P', U)$ but not in $FR(P, P', U)$ exhibits a spectra difference under S that is not associated with exposure of a regression fault. For such an input i, S-spectra-comparison is "imprecise". If $|SR(P, P', U)| = 0$, the degree of imprecision of S with respect to P, P' and U is understood to be 0%, otherwise the degree of imprecision of S with respect to P, P' and U is given by the equation:

$$\frac{|SR(P, P', U) - FR(P, P', U)|}{|SR(P, P', U)|} * 100 \tag{1}$$

Degree of Unsafety. For spectra type S, any input i in U that is in $FR(P, P', U)$ but not in $SR(P, P', U)$ exposes a regression fault that is not associated with a spectra difference of type S. For such an input i, S-spectra-comparison is "unsafe". If $|FR(P, P', U)| = 0$, the degree of unsafety of S with respect to P, P', and U is understood to be 0%, otherwise the degree of imprecision of S with respect to P, P', and U is given by the equation:

$$\frac{|FR(P, P', U) - SR(P, P', U)|}{|FR(P, P', U)|} * 100 \tag{2}$$

Note that the degree of imprecision of S can be determined even though S is not safe. In this respect, the definition of the term "imprecision" used in this paper differs from its definition in areas such as compiler optimization, where safe analyses are required. In the context of maintenance and testing, however, safe analyses are not always necessary – a technique that identifies a sufficiently large subset of some set of facts can be useful, even though it omits some members of that set of facts. However, it is still desirable to know, even for unsafe analyses, the degree to which they identify spurious results. The definition of "imprecision" used in this paper supports this.

Program	Lines of Code	Number of Versions	Input Universe Size	Description
totinfo	431	22	1052	information measure
schedule1	416	7	2650	priority scheduler
schedule2	309	2	2710	priority scheduler
tcas	238	26	1608	altitude separation
printtok1	584	4	4130	lexical analyzer
printtok2	513	7	4115	lexical analyzer
replace	569	20	5542	pattern replacement

Table 3: Experiment 1 subjects.

Note further that when P is correct for input i, and when P' is intended to have the same output for i as P, then i is in $FR(P, P', U)$ if and only if P and P' produce different output for i, or equivalently, if and only if i is in $OPSR(P, P', U)$. That is, when P' is intended to have the same output for i as P, the presence of an output difference between P and P' on i necessarily implies the exposure of a fault in P'. This fact is exploited in the design of the experiments presented in this paper.

3.3 Experiment Instrumentation

3.3.1 Experiment 1 Subjects

In the first experiment seven smaller C programs were used as subjects (see Table 3). Each program has many versions, and each of these versions contains one fault-inducing modification. Each subject program also has a large universe of inputs. These programs, versions, and inputs were assembled by researchers at Siemens Corporate Research for a study of the fault-detection abilities of control- and data-flow coverage criteria [8]. These are referred to as the *Siemens* programs.

The Siemens programs perform a variety of tasks: `tcas` is an aircraft collision avoidance system, `schedule2` and `schedule` are priority schedulers, `tot_info` computes statistics given input data, `print_tokens` and `print_tokens2` are lexical analyzers, and `replace` performs pattern matching and substitution.

The researchers at Siemens sought to study the fault-detecting effectiveness of coverage criteria. Therefore, they created faulty versions of the seven base programs by manually seeding those programs with faults, usually by modifying a single line of code in the program. Their goal was to introduce faults that were as realistic as possible, based on their experience with real programs. Ten people performed the fault seeding, working "mostly without knowledge of each other's work" [8, p. 196].

For each base program, the researchers at Siemens created a large test pool containing possible test cases for the program. To populate these test pools, they first created an initial suite of black-box test cases "according to good testing practices, based on the tester's understanding of the program's functionality and knowledge of special values and boundary points that are easily observable in the code" [8, p. 194], using the *category partition method* and the Siemens Test Specification Language tool [2, 10]. They then augmented this suite with manually-created white-box test cases to ensure that each executable statement, edge, and definition-use pair in the base program or its control-flow graph was exercised by at least 30 test cases. To obtain meaningful results with the seeded versions of the programs, the researchers retained only faults that were "neither too easy nor too hard to detect" [8, p. 196], which they defined as being detectable by at most

Program	Lines of Code	Number of Versions	Universe Size	Description
space	6218	20	13585	interpreter for an ADL

Table 4: Experiment 2 subjects.

350 and at least 3 test cases in the test pool associated with each program.

For the purpose of these experiments, these test pools constitute a "universes of inputs" to these programs.

3.3.2 Experiment 2 Subjects

The second experiment used a single, larger C program developed for the European Space Agency (see Table 4). This program is referred to here as the *Space* program.

The Space program consists of 9564 lines of C code (6218 executable), and functions as an interpreter for an array definition language (ADL). The program reads a file that contains several ADL statements, and checks the contents of the file for adherence to the ADL grammar, and to specific consistency rules. If the ADL file is correct, the Space program outputs a data file containing a list of array elements, positions, and excitations; otherwise the program outputs error messages. The Space program is equipped with a number of faulty versions – each containing a single fault that was discovered during the program's development.

An input universe for the Space program was obtained as follows. An initial pool of 10,000 test cases was obtained from Vokolos and Frankl; they had created the pool for another study by randomly generating test cases [14]. Beginning with this initial pool, the program was instrumented for branch coverage, coverage was measured, and then additional test cases were added to the pool until it contained, for each dynamically executable branch in the control flow graph for the program (excluding those branches that are required to cause execution of `malloc` faults) at least 30 test cases that exercised that branch. This process yielded an input universe of 13,585 inputs.

3.3.3 Tools and Techniques

A variety of tools and techniques were used to compute and record the various types of spectra. For the output spectra (OPS), P and P' were executed on the inputs in U. For the execution trace spectra (ETS), the regression test selection tool `DejaVu` was used [12] to identify the inputs in U that traverse modified statements in the P's.[4] For the branch-hit spectrum (BHS), the branch-count spectrum (BCS), the path-hit spectrum (PHS), the path-count spectrum (PCS), the data-dependence-hit spectrum (DHS), the data-dependence-count spectrum (DCS), and the complete-path spectrum (CPS), various coverage tools from the `Aristotle` analysis system [7] and the `FATE` data-flow testing system [9] were used to record the appropriate entities (i.e., branches, paths, definition-use pairs, complete paths) executed in P and each P'.

[4] In general this approach may identify a superset of the inputs that produce different execution traces; however, in practice it can be determined when the approach incurs imprecision; in all the cases examined in these experiments, the algorithm identified precisely the inputs that produced different execution traces.

3.4 Experiment Design

3.4.1 Variables

Each experiment manipulated a single independent variable, namely, the spectra: OPS, ETS, BHS, BCS, CPS, PHS, PCS, DHS, and DCS. On each run (with P, P', U, and spectra type S), a single dependent variable was measured, namely, the set of inputs in U that revealed S-spectra differences between P and P'.

To measure the fault-revealing behavior of inputs in U, no additional steps were necessary. This is because by construction, for each subject program P (Siemens or Space), the inputs in U that expose faults in P' are exactly the inputs that cause P and P' to produce different outputs. Thus, as discussed in Section 3.2, with these programs, versions, and input universes, $FR(P, P', U) = OPSR(P, P', U)$, and therefore, in these studies, the OPS spectrum serves as a measure of the inputs that expose regression faults in P'.

3.4.2 Design

Each applicable spectra calculation was applied to each (base program, modified version) pair, for each input in the universe for that base program. (Due to limitations in the dataflow analysis tools, DHS and DCS spectra were examined on only the Siemens programs.) This data was used to examine the degrees of imprecision and unsafety of the various spectra, and to compare spectra, using an analysis strategy described in the next subsection.

3.4.3 Analysis Strategy

To examine the relationship between inputs that expose regression faults in P' and inputs that cause P to produce different spectra than P' (objective 1), for each program P and version P', with respect to universe U, and for each spectra type S, the following data was calculated:

1. The degree of imprecision of S with respect to P, P', and U.
2. The degree of unsafety of S with respect to P, P', and U.

To compare spectra (objective 2), for each program P and version P', with respect to universe U, and for each pair of spectra types S_1 and S_2, the following data was calculated:

1. The number of inputs in U that cause spectra differences of type S_1.
2. The number of inputs in U that cause spectra differences of type S_2.
3. The number of inputs in U that cause spectra differences of type S_1 but not of type S_2.
4. The number of inputs in U that cause spectra differences of type S_2 but not of type S_1.

This data was also summarized over the entire set of (program, modified version) pairs, considering each for the entire universe U of inputs (245,087 inputs for the Siemens programs, 271,700 inputs for the Space program). Sections 4.1 and 4.2 present and analyze this data for the two experiments in turn.

3.4.4 Threats to Validity

The primary threats to validity for these experiments are external: these are conditions that limit the ability to generalize the results of the studies to a larger population of subjects. First, the Siemens programs are not large, and it cannot be claimed that they represent a random selection over the population of programs as a whole. Second, the faulty Siemens versions all contain only simple, one- or two-line faults, manually seeded with the intent of simulating "real" faults, but with no data to indicate that they represent a random selection over the population of faults as a whole. These threats can be reduced only by repeated application of the experiment on wider classes of subjects.

Through replication of the experiment on the larger Space program with its real faults, these threats are initially addressed. However, the tools used to collect spectra were not implemented with efficiency in mind, and thus, experimental runs on Space were expensive at 330 hours of machine time per version. Tool improvements are needed to support runs on additional larger subjects.

A second source of threats to validity for this study are internal: these are influences that can affect the dependent variables without the researchers' knowledge. The greatest concern in this study involves instrumentation effects, which can bias results. To control for such effects, two types of cross-checks were utilized: (1) certain results were obtained independently using different instruments at different sites and examined for association; (2) results were examined for adherence to the analytically determined spectra hierarchy relationship. No attempt was made, however, to control for the structure of source programs or for the locality of program changes.

4 Data and Analysis

4.1 Experiment 1

Figure 3 uses boxplots to present the degrees of imprecision and unsafety calculated for each spectra type over the 88 different (program, modified program) pairs of Siemens programs and versions. The vertical axes list degrees of unsafety and imprecision, respectively; the horizontal axes list spectra types. In each boxplot, the dashed line represents the median degree of imprecision or unsafety that occurred for that spectra. The box indicates the interquartile range – the range in which the middle half of the data falls – and also indicates where that data falls with respect to the median. The vertical lines extending above and/or below boxes indicate the percentages at which data above or below the interquartile range fell; however, data points at a distance of greater than 1.5 times the interquartile range are considered outliers, and represented by small circles.

The unsafety data depicted in the figure (left) supports the hypothesis that spectra differences can be expected to occur on inputs that expose faults. Every spectra type demonstrated a median degree of unsafety of 0%. Furthermore, four types of spectra (CPS, PCS, BCS, and DCS) demonstrated a 0% degree of unsafety over their entire first, second, and third quartiles; in other words, on over three quarters of the (program, modified program) pairs, these spectra demonstrated differences on *every* input that exposed a regression fault. For the BHS and PHS spectra, the degrees of unsafety varied more widely than for the other spectra.

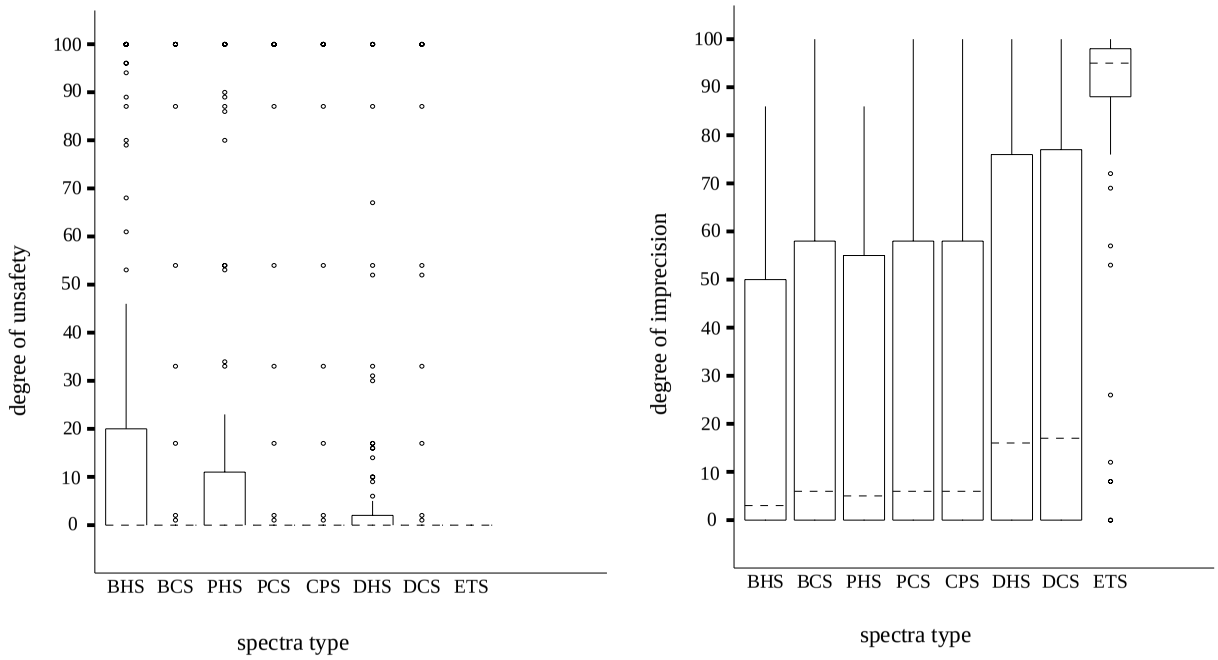

Figure 3: Graphs showing the degrees of unsafety and imprecision of spectra on the Siemens programs.

Finally, all spectra other than ETS displayed occasional extreme (high) unsafety, represented by outliers. Significantly, only ETS was safe: in all cases, all inputs that revealed faults also revealed ETS spectra differences.

The imprecision data presented in Figure 3 (right) illustrates that not all inputs that caused spectra differences also exposed failures: all types of spectra incurred imprecision. However, ETS incurred a much higher median degree of imprecision (95%) than the other types of spectra, for which the median degrees of imprecision ranged from 3% to 17%. Thus, the cost of the safety achieved by ETS, in terms of degree of imprecision, was high. Further, imprecision results for all spectra other than ETS varied widely; exhibiting much greater variance than the unsafety results.

The boxplots also indicate that, in terms of both degrees of imprecision and unsafety, the CPS, PCS, and BCS spectra displayed nearly identical behavior. These results, then, do not support the conjecture that path spectra will be more sensitive indicators of different behavior than branch spectra, where behavior is measured in terms of exposure of regression faults.

Table 5 presents data on the relationships between the various spectra, showing the total number of inputs in the universes that cause each type of spectra difference, and the relations between the sets of inputs identified by each spectra. Column **A** lists the spectra compared; Column **B** lists the total number of inputs that cause spectra differences of type $S1$; Column **C** lists the total number of inputs that cause spectra differences of type $S2$; Column **D** lists the total number of inputs that cause spectra differences of type $S1$ but not of type $S2$; Column **E** lists the total number of inputs that cause spectra differences of type $S2$ but not of type $S1$.

A	B	C	D	E
Spectra (S_1-S_2)	Number of S_1 differences	Number of S_2 differences	S_1 differences not different in S_2	S_2 differences not different in S_1
OPS-BHS	8076	10536	1868	4328
OPS-BCS	8076	17956	469	10349
OPS-PHS	8076	12120	1266	5310
OPS-PCS	8076	17956	469	10349
OPS-DHS	8076	37881	743	30548
OPS-DCS	8076	40372	448	32744
OPS-CPS	8076	17963	469	10356
OPS-ETS	8076	130151	0	122075
BHS-BCS	10536	17956	0	7420
BHS-PHS	10536	12120	0	1584
BHS-PCS	10536	17956	0	7420
BHS-DHS	10536	37881	16	27361
BHS-DCS	10536	40372	12	29848
BHS-CPS	10536	117963	0	7427
BHS-ETS	10536	130151	0	119615
BCS-PHS	17956	12120	5836	0
BCS-PCS	17956	17956	0	0
BCS-DHS	17956	37881	2502	22427
BCS-DCS	17956	40372	12	22428
BCS-CPS	17956	17963	0	7
BCS-ETS	17956	130151	0	112195
PHS-PCS	12120	17956	0	5836
PHS-DHS	12120	37881	29	25790
PHS-DCS	12120	40372	12	28264
PHS-CPS	12120	17963	0	5843
PHS-ETS	12120	130151	0	118031
PCS-DHS	17956	37881	2502	22427
PCS-DCS	17956	40372	12	22428
PCS-CPS	17956	17963	0	7
PCS-ETS	19191	136226	0	117035
DHS-DCS	37881	40372	0	2491
DHS-CPS	37881	17963	22426	2508
DHS-ETS	37881	130151	0	92270
DCS-CPS	40372	17963	22427	18
DCS-ETS	40372	130151	0	89779
CPS-ETS	17963	130151	0	112188

Table 5: Comparison of spectra on the Siemens programs, summarized over all (program, modified version) pairs, considering each for the entire input universe (245,087 data points).

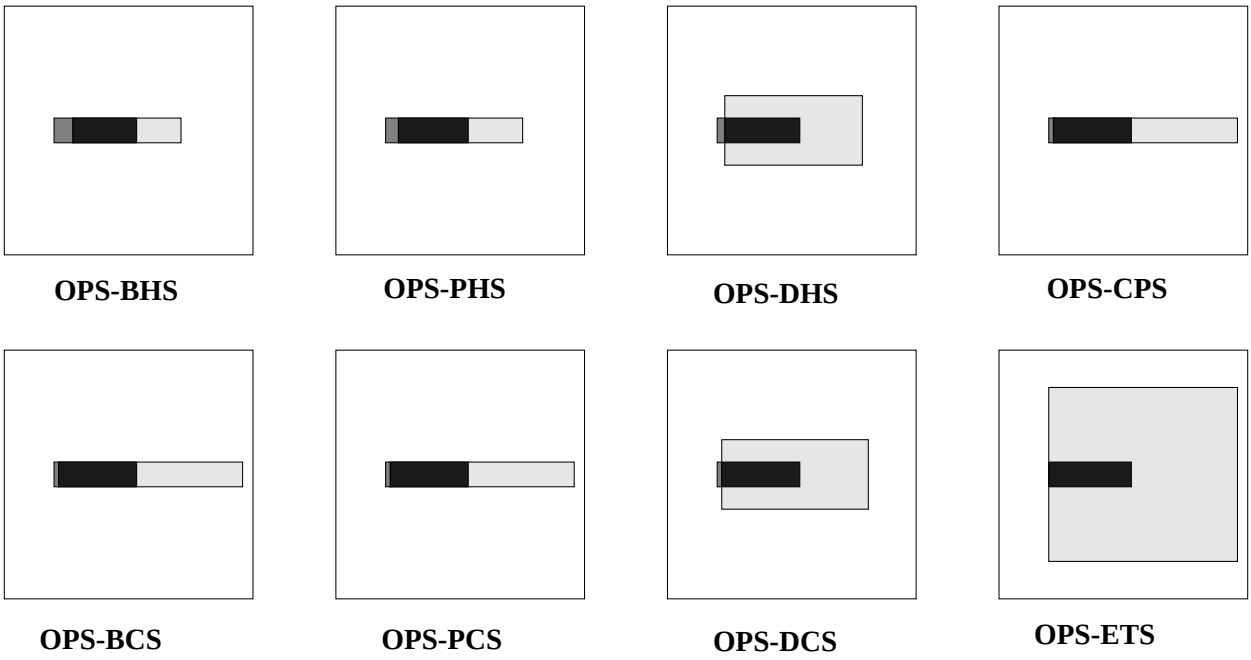

Figure 4: Graphical comparison of OPS with the other spectra, on the Siemens programs.

To aid in interpreting this data, Figure 4 provides a graphical view of the portion of the data in Table 5 that compares the various spectra with OPS, within the context of the entire universe of inputs as applied to all (program, modified program) pairs. In the figure, the eight outer squares represent comparisons of the OPS spectra to the other eight spectra, respectively, as labeled. Each such square represents the entire universe of input points over all (program, modified version) pairs. Within the outer squares, the lightly shaded areas indicate the percentages of input points that caused only XS-spectra differences ($XS \neq OPS$), the medium shaded areas represent the percentages of input points that caused only OPS differences, and the darkly shaded areas represent the percentages of input points under both XS and OPS. Recall that by construction, $FR(P, P', U) = OPSR(P, P', U)$; thus, this graph provides another view of the safety and imprecision of the various spectra types.

It is also possible to compare the sizes of shaded areas in Figure 4 to obtain a qualitative idea of the relationships among the spectra. For example, it is easy to see the relative imprecisions of ETS, DCS, DHS, and CPS in this figure. One observable result: where intuition had prompted the conjecture that DCS and DHS might be better than other spectra types at predicting fault exposing inputs, that intuition was wrong. Instead, DHS and DCS were less precise than all spectra other than ETS, and more unsafe than BCS, PCS, and CPS. Another result: CPS, PCS, and BCS appear quite similar, as do PHS and BHS.

Figure 5 provides a closer view of the relationships between CPS, PCS, PHS, BCS, and BHS that shows, for each of these spectra types, the number of inputs for which spectra differences occurred. Like the imprecision and unsafety results, this figure illustrates the near equivalence, over the programs, modified versions, and input universes considered, of the CPS, PCS, and BCS spectra, as well as the relative closeness

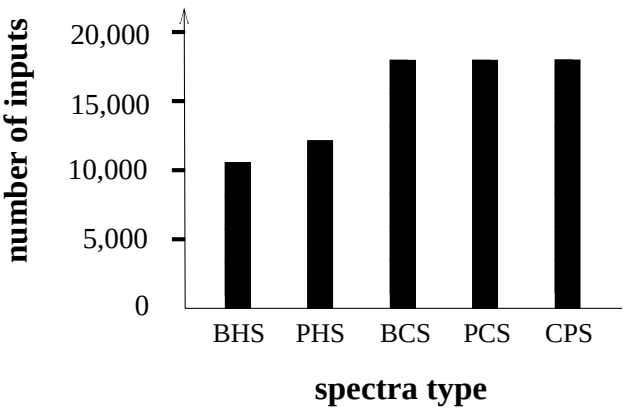

Figure 5: Comparison of CPS, BCS, PCS, PHS, and BHS on the Siemens programs, showing, for each spectra type (horizontal axis), the number of inputs for which spectra differences occurred (vertical axis).

of the PHS and BHS spectra. Finally, where analytical results foretold that neither PHS nor BCS should subsume the other, in this experiment BCS subsumed PHS: BCS spectra differences occurred in 5836 cases in which PHS spectra differences did not occur, but in all cases in which PHS spectra differences occurred, BCS spectra differences also occurred.

These results do not support the conjecture that path spectra will be more sensitive indicators of different behavior than branch spectra. For the 245,087 inputs, PCS was never more sensitive than BCS, and CPS was more sensitive than PCS on only 7 inputs. Essentially, despite the analytical differences between spectra depicted by the subsumption hierarchy, in the cases studied, the CPS, PCS and BCS spectra collapsed into one another, exhibiting nearly equivalent abilities to distinguish differences in program behaviors.

4.2 Experiment 2

Figure 6 shows boxplots, similar to those presented for the Siemens programs in Figure 3, presenting the degrees of unsafety and imprecision of the various spectra types over the 20 different modified versions of the Space program. As the unsafety data (left) indicates, again, spectra differences can be expected to occur on inputs that exposed faults. Again, every type of spectra demonstrated a median degree of unsafety of 0%, with CPS, PCS, and BCS, and DCS demonstrating unsafety greater that 0% only on outliers. For the BHS and PHS spectra, however, the degrees of unsafety demonstrated a much greater range than on the Siemens programs. Again, only ETS was safe.

The imprecision data shown in Figure 6 again indicates that no spectra were perfectly precise. Again, the ETS spectra was the most imprecise with a median degree of imprecision of 91%. The median degree of imprecision for the other spectra (PHS, PCS, BHS, BCS, and CPS) was again 0%; however, in this case, all five of these spectra displayed equivalent ranges of imprecision except for outliers.

Similar to Table 5, Table 6 lists the relationships between the various spectra. Similar to Figure 4, Figure 7 provides a graphical depiction of some of the data in Table 6. In this case, however, the medium shaded areas for OPS-CPS, OPS-PCS, and OPS-BCS are so small (representing only 925 of 271,700 inputs) that

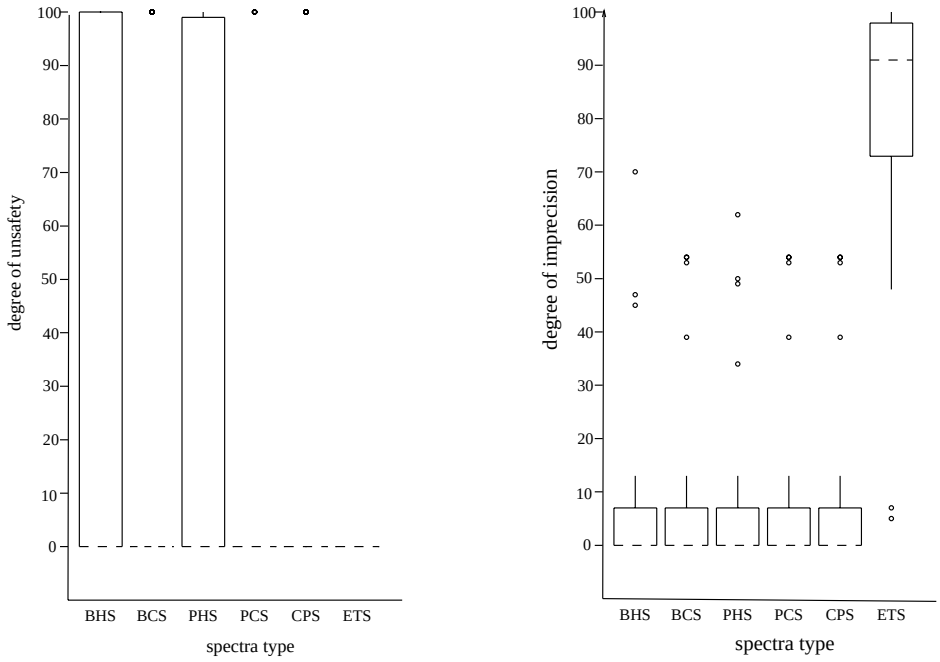

Figure 6: Graphs showing the degrees of unsafety and imprecision of spectra on the Space program.

they are not visible, illustrating the low imprecision of CPS, PCS, and BCS.

Figure 8 provides another perspective on the relationship of BHS, BCS, CPS, PHS, and PCS spectra, similar to that of Figure 5, that displays, for each of those spectra, the number of inputs for which spectral differences existed. With the Siemens programs PCS was found never to be more sensitive than BCS over the 245,087 inputs in the experiment and CPS was found to be more sensitive than PCS on only 7 inputs, whereas with the Space program CPS, BCS, and PCS displayed *identical* sensitivities for all 271,700 inputs.

5 Discussion and Conclusions

This paper has described an empirical investigation of the relationship between spectra differences and the exposure of regression faults, considering the abilities of various spectra types to predict program behavior in terms of such faults, and the relationship between spectra types.

It is important to emphasize that this study has considered only one scenario in which the usefulness of program spectra has been postulated: the scenario in which spectra from a program and modified version, run on the same input, are compared. The results do not provide data applicable to the use of spectra when a single program is run on two slightly different inputs. However, given the degree to which certain spectra types produced nearly equivalent results for our subjects, further empirical study of the relationship between spectra types in the latter scenario would be appropriate.

Further, these studies have focused on just one indicator of program behavior – exposure of regression faults. This indicator is important, and the fact that spectra do correlate with it – at least in one direction – is significant. Whether spectra will correlate with other measures of behavior, such as measures based on

A	B	C	D	E
Spectra (S_1-S_2)	Number of S_1 differences	Number of S_2 differences	S_1 differences not different in S_2	S_2 differences not different in S_1
BHS-BCS	46131	49994	0	3863
BHS-PHS	46131	47655	0	1524
BHS-PCS	46131	49994	0	3863
BHS-CPS	46131	49994	0	3863
BCS-PHS	49994	47655	2339	0
BCS-PCS	49994	49994	0	0
BCS-CPS	49994	49994	0	0
PHS-PCS	47655	49994	0	2339
PHS-CPS	47655	49994	0	2339
PCS-CPS	49994	49994	0	0
OPS-BHS	47308	46131	2923	1746
OPS-BCS	47308	49994	925	3611
OPS-PHS	47308	47655	2235	2582
OPS-PCS	47308	49994	925	3611
OPS-CPS	47308	49994	925	3611
OPS-ETS	47308	201711	0	154403
ETS-BHS	201711	46131	155580	0
ETS-BCS	201711	49994	151717	0
ETS-PHS	201711	47655	154056	0
ETS-PCS	201711	49994	151717	0
ETS-CPS	201711	49994	151717	0

Table 6:
Comparison of spectra on the Space program, summarized over all (program, modified version) pairs, considering each for the entire input universe (271,700 data points).

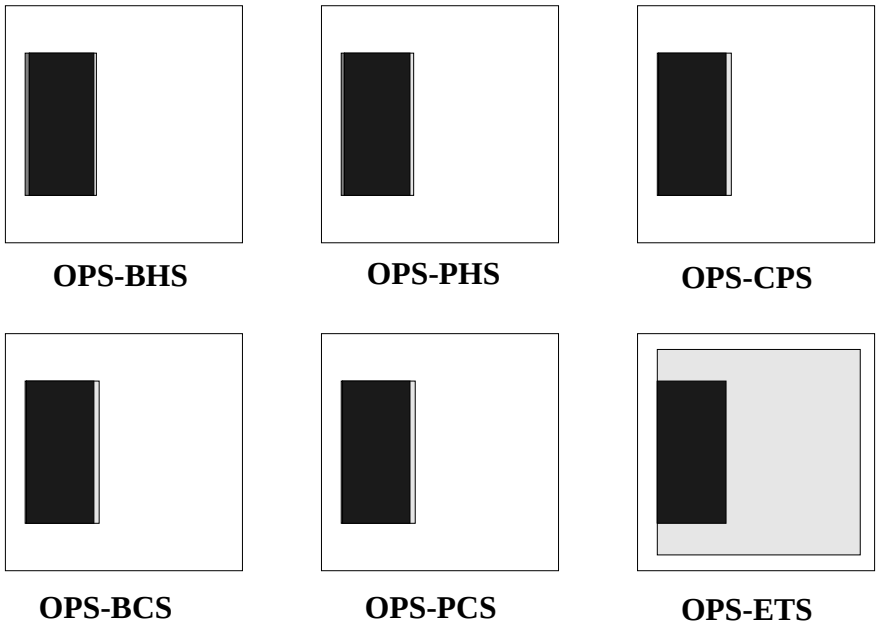

Figure 7: Graphical comparison of *OPS* with the other spectra, for the Space program.

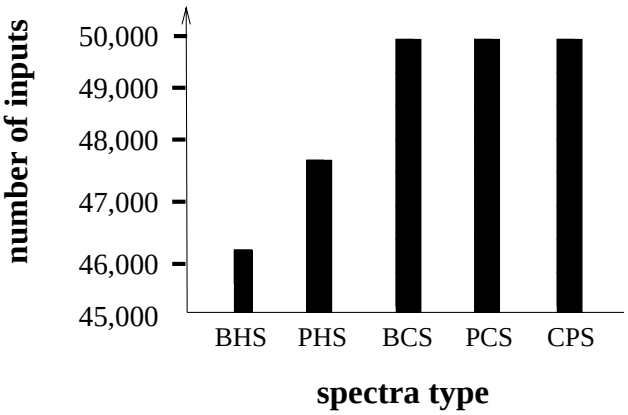

Figure 8: Comparison of CPS, BCS, PCS, PHS, and BHS on the Space program showing, for each spectra type (horizontal axis), the number of inputs for which spectra differences occurred (vertical axis).

sequences of execution states [11], is a subject for future investigation. The comparisons of spectra to each other described in this paper, however, are not restricted to fault-revealing behavior.

Finally, as discussed earlier, there are some threats to validity for this experiment, primarily concerning representativeness of subjects. Additional experimentation with a variety of subjects is necessary.

Keeping the foregoing in mind, however, the results of these experiments support several conclusions. First, although the execution trace spectra emerged as the only spectra to necessarily exhibit differences for inputs that exhibit faults, certain other types of spectra differences (i.e., CPS, PCS, and BCS) also correlate with high frequency (at least in one direction) with fault occurrences. The results suggest that for these spectra types, when failures exist on particular inputs, spectra differences are likely also to exist on those inputs. Moreover, with these spectra types, the degree of imprecision – the frequency with which spectra differences exist but faults do not – is much lower than with the execution trace spectra.

Another conclusion involves the near equivalence of the CPS, PCS, and BCS spectra in terms of their ability to distinguish program behaviors. Program instrumentation has a cost, and in practice that cost must be balanced against potential savings, while also considering the criticality of the application. In general, it would not be cost-effective to collect the complete traces required for CPS spectra; however, if the results of these experiments generalize, PCS and BCS spectra possess power almost equivalent to that of CPS, and may be cost-effective alternatives. Furthermore, estimates suggest that the profiling necessary to collect BCS spectra incurs a 16% run-time overhead whereas the profiling necessary to collect PCS spectra incurs a 30% run-time overhead [5]. In the absence of a gain in sensitivity, use of the PCS spectra instead of the BCS spectra would not be worth the added overhead required to collect the more sensitive spectra.

It is important to note that the experiments reported in this paper do not directly involve techniques for program validation: no specific tools using spectra in software verfication were considered. Nevertheless, this work makes a contribution that should be viewed in a different context from the contributions made by empirical studies of tools. The experiments performed in this work explore fundamental relationships between program behaviors, and techniques for observing those behaviors. By demonstrating that a relatively close

association can exist (in one direction) between certain types of spectral differences and program failure behavior, and by providing data on the possible relationships between types of spectra in practice, the experiments provide a foundation for further research and experimental work on techniques that directly utilize spectra to help with software verification. With this foundation, such research and experimental work can proceed, and hopefully, can yield practical uses of program spectra for software validation.

Acknowledgments

This work was supported in part by a grant from Microsoft Inc., by NSF under NYI Award CCR-9696157 to Ohio State University, ESS Award CCR-9707792 to Ohio State University and Oregon State University, and by CAREER Award CCR-9703108 to Oregon State University. Siemens Corporate Research supplied the Siemens programs. Alberto Pasquini, Phyllis Frankl, and Filip Vokolos provided the `Space` program and many of its test cases.

References

[1] G. Ammons, T. Ball, and J. R. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. *ACM Sigplan Notices*, 32(5):85–96, June 1997.

[2] M. Balcer, W. Hasling, and T. Ostrand. Automatic generation of test scripts from formal test specifications. In *Proc. of the 3rd Symp. on Softw. Testing, Analysis, and Verification*, pages 210–218, December 1989.

[3] T. Ball. On the limit of control-flow analysis for regression testing. In *Proc. of the ACM Int'l. Symp. on Softw. Testing and Analysis*, March 1998.

[4] T. Ball and J. R. Larus. Efficient path profiling. In *Proc. of Micro 96*, pages 46–57, December 1996.

[5] T. Ball, P. Mataga, and M. Sagiv. Edge profiling versus path profiling: The showdown. In *Proc. of 25th ACM Symp. on Prin. of Prog. Lang.*, pages 134–148, Jan. 1998.

[6] Y.F. Chen, D.S. Rosenblum, and K.P. Vo. TestTube: A system for selective regression testing. In *Proc. of the 16th Int'l. Conf. on Softw. Eng.*, pages 211–220, May 1994.

[7] M. J. Harrold and G. Rothermel. Aristotle: A system for research on and development of program-analysis-based tools. Technical Report OSU-CISRC-3/97-TR17, The Ohio State University, March 1997.

[8] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. of the 16th Int'l. on Softw. Eng.*, pages 191–200, May 1994.

[9] J. Lloyd and M.J. Harrold. Implementing an interprocedural dataflow tester using abstract execution. Technical Report 95-111, Clemson University, Clemson, SC, May 1995.

[10] T.J. Ostrand and M.J. Balcer. The category-partition method for specifying and generating functional tests. *Comm. of the ACM*, 31(6), June 1988.

[11] T. Reps, T. Ball, M. Das, and J. Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. *ACM Software Engineering Notes*, 22(6):432–439, November 1997.

[12] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Trans. on Softw. Eng. and Methodology*, 6(2):173–210, April 1997.

[13] G. Rothermel and M.J. Harrold. Analyzing regression test selection techniques. *IEEE Trans. on Softw. Eng.*, 22(8):529–551, August 1996.

[14] F. I. Vokolos and P. G. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Proceedings of the International Conference on Software Maintenance*, pages 44–53, November 1998.

Appendix A: Proof of Theorem 1

Proof:

1. ETS $\rightarrow$ DCS, ETS $\rightarrow$ CPS, ETS $\rightarrow$ OPS .

Given a spectrum $S \in \{\text{DCS}, \text{CPS}, \text{OPS}\}$. Let i be an input that is S-differencing for P and P'. Then, assuming controlled regression testing [13], P' must execute the statement that has changed from P. Thus, i is ETS-differencing, and DCS, CPS, and OPS are subsumed by ETS.

To show that the subsumption is strict, consider a program that has a predicate statement $s : a \leq b$ in P that is changed to $s' : a < b$ in P', and an input i that causes a to be less than b when execution reaches both s and s'. In this case, i is ETS-differencing. However, none of the data-dependencies, the complete path, or the output change for input i, and thus i is not DCS-differencing, CPS-differencing, or OPS-differencing. Thus, equality does not hold, and the subsumption is strict.

2. CPS $\rightarrow$ PCS.

Let i be an input that is PCS-differencing for P and P'. Then there is at least one loop-free path e for P' whose count differs from the count of e in the PCS for P. But this means that there must be a complete path that differs. Thus, i is PCS-differencing and CPS subsumes PCS.

To show that the subsumption is strict, consider the control flow graph of program P, shown on the right in Figure 9. Suppose that, on some input i, the complete path taken through the graph is EabcdfbcdfbceghX. The PCS for P contains paths EabcdfX, EbcdfX, EbcegX, and EhX. Now suppose P' is such that its execution on i produces the complete path EabcdfbcegbcdfhX. Because the complete path is different for P and P' on i, i is CPS-differencing. However, the acyclic paths in P' would be EabcdfX, EbcdfX, EbcegX, and EhX – the same as for P. Thus, equality does not hold, and the subsumption is strict.

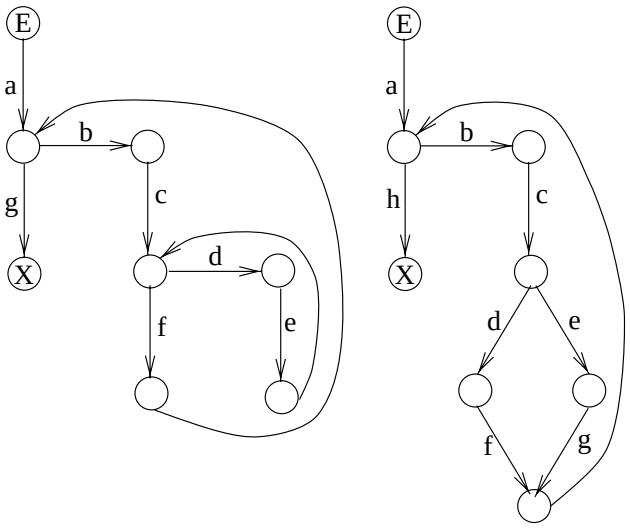

Figure 9: Subgraphs used for proof of Theorem 1.

3. PCS $\rightarrow$ BCS.

Let i be an input that is BCS-differencing for P and P'. Then there is at least one branch e in the BCS for P' whose count differs from the count of e in P's BCS. The paths in PCS are acyclic, so each occurrence of e appears in exactly one path in PCS. Thus, one of the paths containing e has a different count in P and P', and i is PCS-differencing for P and P', and PCS subsumes BCS.

To show that the subsumption is strict, consider the control flow graph on the left in Figure 9. Suppose that, on some input i, the path taken through the graph is EabcdedefbcfgX, and the path taken for P' is EabcdefbcdefgX. Then for both P and P', b is executed twice, g is executed once, d is executed twice, and f is executed twice. For P the paths are EabcdeX, EdeX, EbcfX, EfX, and EgX, whereas for P' the paths are EabcdeX, EfX, EbcdeX, EfX, and EgX. Thus, equality does not hold, and the subsumption is strict.

4. $DCS \rightarrow DHS$, $BCS \rightarrow BHS$ $PCS \rightarrow PHS$.

Let i be an input that is S–hit-differencing for P and P', there exists some entity e (branch, path, data-dependence) that differs in the S-hit spectrum for P and P' on i. Suppose that P's spectrum contains e (branch, path, or data-dependence) that is not contained in P''s spectrum. Then the number of occurrences of e in P's spectrum is at least 1, whereas the number of occurrences of e in P''s spectrum is 0, and thus i is S–count-differencing for P and P'. Thus, S–count subsumes S-hit.

To show that the subsumption is strict, consider an input i that is S–count–differencing, and let P and P' differ only in the number of times that loops are executed. Then, the S–count spectra for P and P' would differ but the S–hit spectra would be the same. Thus, equality does not hold, and the subsumption is strict.

5. PHS $\rightarrow$ BHS.

Let i be an input that is BHS-differencing for P and P'. Suppose that P's BHS contains some branch e that is not in P''s BHS. Then P's PHS must contain at least one path containing e, whereas P''s PHS contains none of the paths containing e. Thus, i is PHS-differencing for P and P', and PHS subsumes BHS.

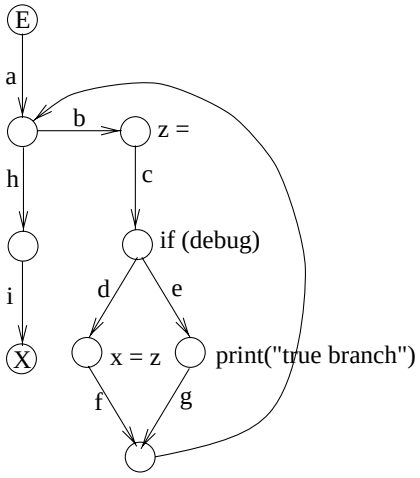

Figure 10: Subgraph used for proof of Theorem 1.

To show that the subsumption is strict, consider the control flow graph of program P, shown on the left in Figure 9. Suppose that, on some input i, the actual path taken through the graph is EabcfbcfgX. The BHS for P contains branches b, f, and g; the PHS contains paths EabcfX, EbcfX, and EgX. Now suppose that P' is such that it executes the loop only one time. Then the BHS for P' contains b, f, and g – the same as the BHS for P. However, the PHS for P' has EabcfX and EgX. Thus, equality does not hold, and the subsumption is strict.

6. OPS is incomparable with CPS, PCS, PHS, BCS, and BHS.

6a. Consider OPS with CPS. Let P be a program with computation $x = a + b$, and let P' be a modified version in which $x = a + b$ is changed to $x = a - b$, such that neither $x = a + b$ nor $x = a - b$ affects the flow of control in P or P'. In this case i can be OPS-differencing, but it is not CPS-differencing for P, P', and i. Thus, CPS does not subsume OPS, and thus none of CPS, PCS, PHS, BCS, or BHS subsumes OPS.

6b. Now consider BHS and OPS,and a program P in which a section of code is replaced by a semantically equivalent piece of code that has a different control flow structure to get P', let i be an input that produces the same output for P and P' (of course all i should do this) but traverses different branches in P'. In this case, i is BHS-differencing but not OPS-differencing. Thus, BHS does not subsume OPS, and thus none of CPS, PCS, PHS, BCS, or BHS subsumes OPS.

By 6a and 6b, OPS is incomparable to CPS, PCS, PHS, BCS, and BHS.

7. DCS and DHS are incomparable with CPS, PCS, PHS, BCS, and BHS

7a. Consider DHS with CPS. Let P contain a statement $s : a = B + C$, and let P' contain a modified version $s' : a = b + 1$. Let i be an input that produces the same complete path in P and P'. Clearly i produces a difference in the definition-use pairs in P and P', and thus, i is DHS-differencing for P, P', and i. However, because the path executed in P and P' for i is the same, i is not CPS-differencing for P, P', and i. Thus, CPS does not subsume DHS, and thus none of CPS, PCS, PHS, BCS, or BHS subsumes DHS or DCS.

7b. Now consider BHS and DCS, and the partial control flow graph for shown in Figure 10. Assume that this

graph models both P and P', and that the only difference between the two is that the conditional statement associated with the node that is the target of edge a has been changed. Let i be an input such that when P is executed with i, it takes the path EabcdfbceghiX and when P' is executed with i, it takes the path EabcdfhiX. In this case, i is BHS-differencing for P and P' but not DCS-differencing for P and P'. Thus, DCS does not subsume BHS, and thus neither DCS nor DHS subsumes CPS, PCS, PHS, BCS, or BHS.
By 7a and 7b, DCS and DHS are incomparable with CPS, PCS, PHS, BCS, or BHS.
8. OPS is incomparable with DCS and DHS.
8a. Consider OPS and DCS, and the example given in part 6a above. If i is such that it produces a different output on P and P', then i is OPS-differencing, but not DCS-differencing for P and P'. Thus, neither DCS nor DHS subsumes OPS.
8b. Now consider DHS and OPS, and the example given in 7a above with i such that P and P' produce the same output when executed with i as input. Then i is DCS-differencing, but not OPS-differencing for P, P', and i. Thus, OPS does not subsume DCS, and thus it does not subsume DHS.
By 8a and 8b, OPS is incomparable with both DCS and DHS.
9. BCS and PHS are incomparable.
9a. Consider the partial control flow graphs on the left of Figure 9, and the case where input i causes execution of the path EabcfbcfbcfX when used as input for P but causes execution of the path EabcfbcfX when used as input for P'. The number of times branch b is executed differs in P and P', and thus i is BCS-differencing for P and P'. The paths that are executed in P and P', however, are the same – EabcfX, EbcfX, and EgX in both cases – and thus i is not PHS-differencing for P and P'. Thus PHS does not subsume BCS.
9b. Now consider the partial control flow graph on the right of Figure 9, and i such that when P is executed with i, it takes the path EabcdfbceghX and when P' is executed with i, it takes the path EabcegbcdfhX. Clearly, i is PHS-differencing, but not BCS-differencing, for P and P'. Thus, BCS does not subsume PHS.
From 9a and 9b, BCS and PHS are incomparable.
This completes the proof. □