Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
00 (2023) 1–64
~~Journal of~~
Systems
and
~~Software~~

What Makes Test Programs Similar in Microservices Applications?

Emanuele De Angelis[a,*], Guglielmo De Angelis[a,*], Alessandro Pellegrini[b,a,*], Maurizio Proietti[a,*]

[a]*IASI-CNR, Rome, Italy*
[b]*University of Rome Tor Vergata, Rome, Italy*

Abstract

The emergence of microservice architecture calls for novel methodologies and technological frameworks that support the design, development, and maintenance of applications structured according to this new architectural style. In this paper, we consider the issue of designing suitable strategies for the governance of testing activities within the microservices paradigm. We focus on the problem of discovering implicit relations between test programs that help to avoid re-running all the available test suites each time one of its constituents evolves. We propose a dynamic analysis technique and its supporting framework that collects information about the invocations of local and remote APIs. Information on test program execution is obtained in two ways: instrumenting the test program code or running a symbolic execution engine. The extracted information is processed by a rule-based automated reasoning engine, which infers implicit similarities among test programs. We show that our analysis can be used to support the reduction of test suites. The proposed approach has been validated against two real-world microservice applications.

Keywords: software testing, microservices architecture, test program similarity, symbolic execution, program instrumentation, automated reasoning.

1. Introduction

The microservices architecture style consists in building a software application as a collection of distributed software units, each abiding by the single responsibility principle [1]. The functionalities offered by a microservice are supposed to be contained within clearly defined boundaries, encapsulating the implementation of atomic features in the considered domain [2]. Also, the microservices architecture principles suggest a strong control of the coupling among software units, advocating the adoption of design solutions that mitigate the impact of the evolution of each microservice. In other words, going through the various life-cycle phases of each microservice (i.e., its design, development, deployment, or update) should require minimal (or even zero) coordination effort with the others, possibly limited to immediate dependencies.

Both the technical and the managerial independence of microservices should cope with a dynamic scenario for the development and maintenance of applications built within this paradigm: the evolution of one or more constituents could occur according to several governance schemata, opening to different degrees of challenges about the resulting system [3]. In order to take full advantage of this architectural style, novel methodologies and new technological frameworks are needed for designing, developing, and maintaining microservices applications. Also, testing activities demand appropriate strategies and tools covering each test phase: from unit testing to integration, and contract testing, up to end-to-end testing [4]. In addition, the continuous evolution of any of the constituents suggests the establishment of procedures and resources ascertaining that changes have not caused novel and undesired issues. Across the different stages of testing activities, regression testing [5] aims to guarantee that the changes introduced in a software module do not harm its behaviour or the one exposed by the whole software system.

In the case of governance of regression testing activities, several classes of approaches aim at preventing the *retest-all* strategy by: i) skipping redundant test cases from the test suite [6], or ii) selecting some test cases [7], or iii) prioritising those expected to yield earlier fault detection [8], [9]. However, in most

[*]Corresponding author
Email addresses: `emanuele.deangelis@iasi.cnr.it` (Emanuele De Angelis), `guglielmo.deangelis@iasi.cnr.it` (Guglielmo De Angelis), `a.pellegrini@ing.uniroma2.it` (Alessandro Pellegrini), `maurizio.proietti@iasi.cnr.it` (Maurizio Proietti)

cases, these approaches require some knowledge about the considered set of microservices, their immediate dependencies, and their possible interactions. Unfortunately, the lack of detailed specifications for the considered microservices and, in some cases, the unavailability of the source code could hamper the direct application of such testing techniques [10].

In addition, regression testing activities have to cope with the maintenance and the evolution of the regression test suites [11]: augmenting their significance by deriving new test cases from existing ones or by inferring a better understanding of the considered software system by leveraging pieces of evidence from the test cases. In this respect, the observation of the actual interactions among microservices instances can be exploited as a means of contributing to the evolution of regression testing test suite [12]. Also, test cases have been proposed as viable solutions for checking compliance of contracts across service releases [13].

This work contributes to the governance of regression testing in the specific context of microservices applications. One relevant piece of information often useful when designing regression testing strategies is the similarity between test cases. For example, test cases could be considered similar if they include the same activities but focus on a different testing strategy; if they target the same testing goal and strategy but use different test data; or if parametric tests have significant overlaps for some values of the parameters. Inferring such relationships is a complex task in the general case, as they strongly depend on the specific nature of the considered software system (e.g., application domain, referred architectural style, adopted technologies). As detailed in the following, this work leverages the specificity of the microservices paradigm in order to structure similarity information retrieval procedures that enable reasoning about test program similarities. Then, the knowledge of test case similarities allows the design of flexible regression testing strategies and policies, which avoid rerunning all test programs in an order fixed in advance.

Specifically, this work assumes that a set of test programs for a given microservices application is available because they are shipped with the microservices, or some system integrator made them available (e.g., contract tests for microservices that are commonly used together), or the integrator of the overall application provides them. Then, we propose a dynamic analysis of the given test programs to discover suitable similarities among them. Our analysis relies on both instrumented and symbolic execution techniques [14] to gather information about the behaviour of a test program and the interactions it establishes among the microservices in the application under test. While the instrumented execution allows us to collect the trace of one concrete execution quickly, the symbolic approach allows the exploration of sets of concrete executions and allows us to handle parametric tests naturally. The information extracted is processed using logic-based reasoning techniques [15] in order to establish similarity relations.

In order to evaluate our approach, we have implemented our analysis and reasoning techniques on a tool, called Hyperion, which is publicly available as open-source software.[1] Then, we have worked out two real-world case studies and we have shown that Hyperion is indeed capable of discovering similarities between sets of test programs, according to the various criteria we have defined. Our results also show that the similarities discovered can be used profitably, for example, to reduce the test case suite, and thus our approach has good potential for use in regression test optimisation. However, providing full-blown strategies for regression test optimisation, e.g., test case prioritisation, minimisation or selection, is beyond the scope of this paper.

This paper builds on the results presented in previous papers [16, 17] and extends them in several ways. In particular, a first extension concerns the definition of a new set of similarity metrics: the work in [16] does not concern similarity metrics, while the work in [17] only refers to a collection of set-based similarity criteria; this work also defines and implements several sequence-based similarity criteria. Furthermore, both previous works only refer to scenarios where the similarities are computed starting from the symbolic execution of the test programs; in this work, we have extended both the methodology and its supporting framework to evaluate similarities starting from the concrete execution of the test programs. Finally, this work also enhances the validation of the proposed approach by performing, as mentioned above, an empirical evaluation on two popular open-source applications designed according to the microservices architectural style.

The rest of the paper is organised as follows. Section 2 provides some background about the main techniques used in this work. In Section 3 we present our overall approach to extracting relations among test programs from their concrete or symbolic executions. In Section 4 we describe the technique used to extract information from test programs, while in Section 5 we introduce the rules to determine their similarity. Section 6 describes the validation methodology that we have applied in the empirical study reported in Section 7. In Section 8 we comment on the threats to the validity of the empirical evaluation of our technique. Section 9 discusses related work and, finally, Section 10 draws the conclusions of this work.

2. Background

This section recalls some background notions about symbolic execution, software instrumentation, and logic programming, which are the core assets for the proposed contribution.

2.1. Symbolic Execution

Symbolic execution [18] is an established technique in automated software testing [14] for exploring program executions in search for runs that lead to error states, that is, states where some specified conditions are violated. Unlike concrete execution, where a program is run on a specific input, and a single

[1]Source code available at `http://saks.iasi.cnr.it/tools/hyperion`.

```
1   public void foo(int a, int b) {
2       int x = 0, y = 1;
3       if(a > 0) {
4           x = 2 * y;
5           if(b < 0)
6               y = a - b;
7       }
8       assert(x - y != 0);
9   }
```

Figure 1: Which values of a and b make the `assert()` fail?

control flow path is explored, the basic idea of symbolic execution is to allow *symbolic variables*, that is, variables that take on symbolic values, besides concrete values. The use of symbolic variables allows the simultaneous exploration of multiple paths a program can take under different inputs. Every time that some condition is checked against a symbolic variable, a *branch* is taken and alternative control flows are maintained simultaneously by the *symbolic execution engine*.

For clarity, let us consider the example code snippet in Figure 1. Symbolic execution can effectively determine which inputs make the final assertion fail without having to enumerate the whole set of possible input values. Indeed, by relying on symbolic variables, one could reason on *classes of inputs*.

Every time a conditional branch instruction is symbolically executed, the symbolic execution engine creates a "snapshot" of the execution context up to that point. This snapshot can be used to backtrack to a previous execution state and restart the execution to explore alternative execution possibilities. Therefore, the overall symbolic execution of the program can be represented as a tree, where each conditional branch instruction generates two additional sub-trees describing the possible outcomes of the comparison.

The symbolic execution engine that supports the symbolic execution can be regarded as an abstract machine, which maintains a state represented by the triple $\langle insn, \sigma, \pi \rangle$, where:

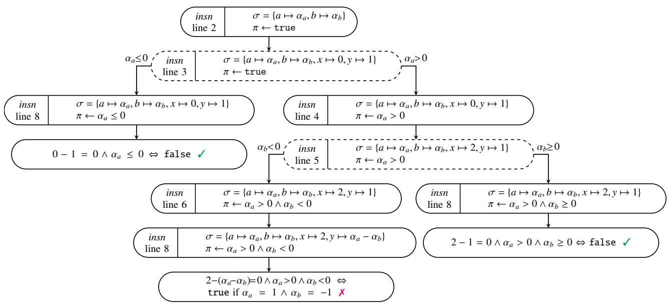
insn is the program point that has been reached during the symbolic execution of the program;

σ is a symbolic memory store, associating variables with either expressions over concrete values or symbolic values α_i;

π is a first-order logic formula—the so-called *path condition*—i.e., a formula that expresses a set of constraints on the symbols α_i built during the execution of the branches observed up to *insn*.

Any branch instruction executed on a symbolic variable updates the path condition π, while assignments update the symbolic store σ. A Satisfiability Modulo Theories (SMT) solver checks whether there are any violations of the constraints along each explored path and if the path itself is feasible. The tree associated with the code example in Figure 1 is shown in Figure 2. In this figure, we can observe that multiple symbolic states are traversed to reach leaf states. In the leaves, the symbolic store is used to check whether or not the `assert` condition fails.

The fact that symbolic execution traverses *all* states $\langle insn, \sigma, \pi \rangle$ up to a certain program point opens up for further exploitation



insn line 2 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b\}$, $\pi \leftarrow$ true
$\alpha_a \leq 0$
insn line 3 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 0, y \mapsto 1\}$, $\pi \leftarrow$ true
$\alpha_a > 0$
insn line 8 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 0, y \mapsto 1\}$, $\pi \leftarrow \alpha_a \leq 0$
insn line 4 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 0, y \mapsto 1\}$, $\pi \leftarrow \alpha_a > 0$
$0 - 1 = 0 \wedge \alpha_a \leq 0 \Leftrightarrow$ false ✓
$\alpha_b < 0$
insn line 5 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 2, y \mapsto 1\}$, $\pi \leftarrow \alpha_a > 0$
$\alpha_b \geq 0$
insn line 6 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 2, y \mapsto 1\}$, $\pi \leftarrow \alpha_a > 0 \wedge \alpha_b < 0$
insn line 8 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 2, y \mapsto 1\}$, $\pi \leftarrow \alpha_a > 0 \wedge \alpha_b \geq 0$
insn line 8 | $\sigma = \{a \mapsto \alpha_a, b \mapsto \alpha_b, x \mapsto 2, y \mapsto \alpha_a - \alpha_b\}$, $\pi \leftarrow \alpha_a > 0 \wedge \alpha_b < 0$
$2 - 1 = 0 \wedge \alpha_a > 0 \wedge \alpha_b \geq 0 \Leftrightarrow$ false ✓
$2-(\alpha_a-\alpha_b)=0 \wedge \alpha_a>0 \wedge \alpha_b<0 \Leftrightarrow$ true if $\alpha_a = 1 \wedge \alpha_b = -1$ ✗

Figure 2: Symbolic execution tree of the example program in Figure 1. Dashed boxes correspond to states in which a branch is taken. The leaf node marked with a ✗ is associated with a terminal state which violates the `assert()` in the example program.

of this technique, which is the basis of this work. Indeed, while the original goal of symbolic execution was to explore all the possible execution paths to determine what inputs to a test program might generate an error condition, since we can *observe* all the states during the symbolic execution, we can extract information about *all* possible activities carried out by a *parametric* test program. This can also be done in terms of the parts of the system under test (i.e., SUT), which are exercised by the test program itself, independently of the concrete values passed as input to the test by the developers.

2.2. *Software Instrumentation*

Another technique typically employed to support code coverage analysis is software instrumentation [19]. In Java, code instrumentation is typically carried out either at the source code level or the bytecode level. For our purposes, we focus on the latter.

Bytecode instrumentation techniques can be broadly classified as *static*, or *dynamic* approaches. In static bytecode instrumentation, all instrumentation code (e.g., software probes to inspect the application's behaviour) is inserted in the application before the program starts execution. Typically, this approach causes less runtime overhead, as all the bytecode mangling is performed before the process is launched. The major drawback is that dynamically generated or loaded code cannot be instrumented.

Conversely, dynamic bytecode instrumentation is interleaved with the program's execution under instrumentation. Typically, this approach relies on an *instrumentation agent* that is invoked every time a new class is loaded. The agent can analyze the bytecode of the loaded class and augment the loaded bytecode with instrumentation code. The major benefit of this approach is that multiple agents can coexist, and typical support offered by the Java Virtual Machine allows all the available agents to chain the bytecode modification. Usually, this induces higher overhead (mainly at program startup) and may affect measurements due to the runtime instrumentation process. Dynamic instrumentation also has the additional benefit that only those classes being actually loaded are instrumented, while static instrumentation requires processing all the classes, even though some may not be executed in a given scenario.

Typically, both static and dynamic instrumentation rely on *bytecode engineering libraries*, such as ASM [20], or Javassist [21].

2.3. Logic Programming

We recall the basic concepts of logic programming that we will use to reason about the similarity of test programs.

The logic programming syntax builds upon *terms* and *statements*. A term is either a *variable*, a *constant*, or a *compound term*. A statement is either a *fact*, a *rule*, or a *query*.

A fact is used to state a relation among objects, and is represented as an atomic formula, that is, a predicate symbol of arity $n \geq 0$ applied to n terms. A rule is an implication of the form `head :- body`, where: (i) `head` is an atomic formula representing the conclusion of the implication, (ii) '`:-`' denotes the (reverse) implication symbol $\leftarrow$, and (iii) `body` is a conjunction of atomic formulas representing the premise of the implication. A logic program consists of a set of facts and rules. A query is used to ask whether a relation among objects holds. Syntactically, queries are atomic formulas, like facts, but the usage context can distinguish them. Any answer to a query with free variables provides values for the variables that make the query a logical consequence of the logic program.

We use the Prolog programming language as a concrete realisation of the logic programming paradigm and the SWI-Prolog system [15] as the reference implementation of Prolog. In presenting logic programs, we will follow the usual conventions of Prolog: variables begin with an uppercase letter, while constant, function, and predicate names begin with a lowercase letter.

3. Overall Approach

Often Quality Assurance (QA) teams agree on policies and strategies for regression testing within a shared governance framework [3]. Such a framework supports the decision process during the testing campaigns, for example, by guiding the activities that could support the root-cause analysis of issues that have been spotted.

The enforcement of specific decisions can be either planned in advance of the regression testing activities (e.g., test suite reduction, test case selection/prioritisation) or online through a test case orchestrator [22] that dynamically makes decisions on how the regression testing process proceeds by taking into consideration the actual outcomes resulting from the test cases executions. In both cases, the role of test suites dependencies/similarities can foster the definition of parallel, sequential, or alternative flows of test cases to be executed [3].

Different factors can lead to establishing dependencies across regression test cases. On the one hand, members of the QA team or even software developers could declare them either in the software specifications or in the configurations of the referred build automation frameworks. On the other hand, implicit similarities can also be drawn from available software artefacts (e.g., test programs) using some (semi-)automatic mining procedures. In this article, we focus on this latter scenario.

The microservices architectural style suggests the design of highly modular applications, where the responsibilities of each microservice, its boundaries, and its interconnections are clearly identifiable [2]. Given a collection of integration test cases, their elements could be considered similar if they concern the same set of microservices. In addition, all the unitary tests for a given microservice ms_i could also be considered related to integration tests that involve ms_i: a failing integration test should also prompt to check whether any unit regression has occurred in the microservices it refers to. Criteria of this kind have been initially introduced in [3] where the discussion also covered dependencies that could be established at all the test levels (e.g., contracts, end-to-end). In the following, test programs for microservices applications are considered "similar" if they:

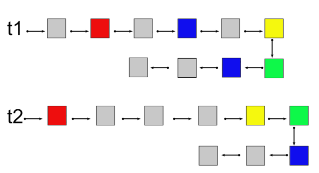
1. involve the same microservice instance, or they connect to the same remote API;
2. locally activate overlapping APIs (i.e., they refer similar local modules/libraries).

Microservices are distributed components whose interactions occur across some abstraction of the network interface and, in most cases, abide by the REST architectural style [1]. Test programs opening connections against the same remote APIs act as test drivers for the same type of microservices or, in some cases, among the same instances. Such connections to remote APIs can be directly coded in the test program employing basic frameworks that provide functionalities for accessing resources via HTTP (e.g., the HTTP Clients in the JDK or Apache libraries). Also, their implementation could be mediated by means of some structured application framework (e.g., Spring[2] or Postman[3]), or even mediated by means of some local libraries automatically generated starting from the remote APIs specifications (e.g., the client SDKs generated by means of Swagger Codegen[4]). This last technological solution opens the possibility of looking for similarities among those test programs that locally activate overlapping APIs. In addition, item 2 is also considered useful when looking for test programs that converge onto a cohesive set of activities: for testing purposes, but also for the configuration of the test environment or their referred assertions.

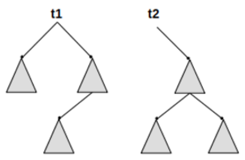
Our overall approach is depicted in Figure 3. Identifying similarities among test programs is guided by an automatic analysis of their implementations and executions, which does not rely on any specification of the tests. The analysis procedure assumes that test programs are clearly identifiable from the rest of the source code or compiled classes, for example, through explicit JUnit annotations. Also, it is based on two different modes of test program execution: concrete or symbolic.

When the former mode is enabled, the analysis procedure runs each available test program and, through program instrumentation, automatically records the API that the test program

[2]see: `https://spring.io/microservices`
[3]see: `https://www.postman.com`
[4]see: `https://swagger.io/tools/swagger-codegen/`



(a) Concrete Execution Traces



(b) Symbolic Execution Tree

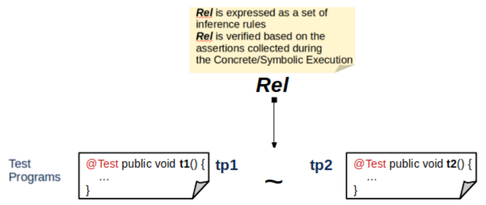

Rel is expressed as a set of inference rules
Rel is verified based on the assertions collected during the Concrete/Symbolic Execution
Rel
Test Programs
@Test public void t1() { ... }
tp1
~
tp2
@Test public void t2() { ... }
(c) Analysis of Test Programs

Figure 3: Overall Approach.

locally activates. This mode aims to extract implicit dependencies across test programs that are actually coded in their implementations or due to specific values associated with the test program's arguments (see Figure 3a). By enabling the latter, the analysis procedure still runs the available test programs, but it can also be configured to automatically switch their execution to symbolic processing (see Figure 3b). The aim of this mode is twofold: (i) to carve test data by exploring admissible but not explicitly coded executions that are subsumed by the test program, and (ii) to exercise (parametric) test programs independently of their arguments.

The outcome of the execution phase is a knowledge base consisting of facts representing the configurations reached by either concrete or symbolic execution. Then, in a subsequent phase, the knowledge base is analyzed to reveal existing similarities among test programs (see Figure 3c). Specifically, this second phase builds upon a given set of inference rules that define similarity criteria among test programs, and whose evaluation is performed by querying the knowledge base generated during the execution phase.

An initial set of inference rules has been investigated in [16] and then extended in [17]. Even though the proposed approaches are modular enough to cover additional dependency criteria, in this work, we have further improved the existing inference rules and validated them by means of broader experimental activities that cover outcomes of both the concrete and symbolic modes of the analysis procedure.

4. Carving Behavioral Features from Test Programs

This section details the methodology used to represent the information carved from test programs and the techniques we have explored to generate such data, namely symbolic and concrete execution. We also discuss our reference implementations.

4.1. Program Traces Representation

In order to identify the behavioural features of test programs for subsequent similarity characterisation, we must first be able to represent the execution trace of these programs.

```java
public void f() {
  for(int i = 0; i < 5; i++)
     g();
}
```

Listing 1: A function with calls within a **for** loop.

```
invokes(f, [1], f, g)
invokes(f, [1], f, g)
invokes(f, [1], f, g)
invokes(f, [1], f, g)
invokes(f, [1], f, g)
```

Listing 2: Sequence of `invokes` generated from Listing 1.

Our representation is based on a set of Prolog facts that represent an invocation of a particular method within a specific class from a certain caller, then enabling suitable rule-based reasoning techniques. The simplest format of these facts is the following:

```
invokes(TestProgram, Caller, Callee)
```

where `invokes` is the predicate name, `TestProgram` is a unique identifier of the test program in the currently-analyzed test suite, `Caller` is the invoking method, and `Callee` is the invoked method.

With this format, we are not explicitly considering the twofold nature of carving methods. In particular, when generating facts from a symbolic execution, we must keep track of the point where one specific invocation was observed in the symbolic execution tree. Conversely, we only refer to a single execution trace in a concrete execution. In other words, the facts must maintain the information that, in symbolic execution, they describe the *possibility* that, for specific concrete inputs to the test program, a particular method invocation could be materialized in a concrete execution.

To this end, we introduce a list of *branching points*, which are a linear representation of a path in the symbolic execution tree. These facts thus become:

```
invokes(TestProgram, BranchingPointList, Caller, Callee)
```

Of course, in the case of a concrete execution, `BranchingPointList` will be set to a placeholder value identifying that only a single execution trace has been generated, meaning that no branching was observed nor relevant.

To understand whether this format is sufficient to represent an execution trace to carry out a behavioural analysis, let us now consider the example code snippet in Listing 1. Here, we find repeated invocations to `g()`, which will in turn generate multiple `invokes` facts, as shown in Listing 2—`[1]` is used as `BranchingPointList` to indicate that the example in the listing refers to a single execution. The `invokes` in the figure are exactly the same, but one could argue that every single `invokes` fact generated by a call to `g()` is a different incarnation and should be therefore considered different. To enforce this difference, we extend the form of the `invokes` facts as follows:

```
invokes(TestProgram, BranchingPointList, SeqNum, Caller, Callee)
```

```
public void a(int count) {
    b();
    if(count > 0)
        a(0);
    c();
}
```

Listing 3: A recursive function.

```
public void a(int count) {
    if(count == 0)
        return;
    b();
    a(0);
    b();
    c();
    c();
}
```

Listing 4: A fragment generating a sequence of `invokes` equivalent to that of Listing 3.

where `SeqNum` is a monotonic counter which is incremented every time that an `invokes` fact is generated. Therefore, every invocation of `g()` in the example in Listing 1 will bear a different value for `SeqNum`, thus allowing us to disambiguate invocations within iterations.

Let us now consider the example shown in Listing 3. Here, a different set of methods is invoked depending on the (either concrete or symbolic) value of the method parameter `count`. If the example program is invoked as `a(1)`, a sequence of facts corresponding to the invocations of `b()`, `a(0)`, `b()`, `c()`, `c()` will be generated, all appearing as being called from `a()`. Here, the problem is that the same sequence of facts could also be generated by the example program in Listing 4. In particular, both programs would generate the sequence of `invokes` shown in Listing 5. The two programs are inherently different though, and should not be described by the very same sequence of `invokes`. While the example deals with a recursive invocation, we note that the same problem might arise with repeated invocations of the same method from the same caller.

This anomaly stems from two different issues. First, the `invokes` fact as defined above cannot distinguish between different invocation contexts. Second, the invocations are different because they come from two different places in the source programs. To overcome this limitation, we enhance the form of the `invokes` facts as follows:

```
invokes(TestProgram, BranchingPointList, SeqNum, Caller,
     ProgramPoint, FrameEpoch, Callee)
```

where `ProgramPoint` is a unique identifier of the location of the method call in the original program (for example its line number in the original source file), and `FrameEpoch` is an additional monotonic counter handled as follows. Every time a method invocation occurs in the symbolic execution, this counter is incremented. The new value is then pushed onto a stack. Every `invokes` fact is annotated with the value associated with the caller, i.e., the second-to-top element on the stack. Every time a return instruction is symbolically executed, we pop the top element from the stack. In this way, recursive or repeated

```
invokes(a, [1], 1, a, b)
invokes(a, [1], 2, a, a)
invokes(a, [1], 3, a, b)
invokes(a, [1], 3, a, c)
invokes(a, [1], 3, a, c)
```

Listing 5: A sequence of `invokes` that can be associated with both Listing 3 and Listing 4.

```
Invokes for Listing 3:
        invokes(a, [1], 1, a, 2, 1, b)
        invokes(a, [1], 2, a, 4, 1, a)
        invokes(a, [1], 3, a, 2, 2, b)
        invokes(a, [1], 4, a, 5, 2, c)
        invokes(a, [1], 5, a, 5, 1, c)

Invokes for Listing 4:
        invokes(a, [1], 1, a, 4, 1, b)
        invokes(a, [1], 2, a, 5, 1, a)
        invokes(a, [1], 3, a, 6, 1, b)
        invokes(a, [1], 4, a, 7, 1, c)
        invokes(a, [1], 5, a, 7, 1, c)
```

Listing 6: Invokes discriminating Listing 3 and Listing 4.

invocations will bring a different frame epoch for every called method. This construction allows us to mimic the behaviour of stack frames employed by computer architectures for the same purpose. Additionally, if a method is invoked from a different location in the original source, it will have a different `ProgramPoint` value. The resulting (different) invokes for the snippets Listing 3 and Listing 4 in are reported in Listing 6.

Finally, we might consider two invocations to the same method as similar if they have the same set of parameters—in the case of symbolic execution, the parameters might be symbolic as well. We note that in the microservices scenario we target, we are not interested in argument values (except for strings) but rather in parameter types. Similarly, discriminating whether two invocations are the same might entail considering also the symbolic path condition. To this end, the final incarnation of the `invokes` becomes:

```
invokes(TestProgram, BranchingPointList, SeqNum, Caller,
     ProgramPoint, FrameEpoch, PathCondition, Callee, Parameters)
```

Similarly to the `BranchingPoint` case, `PathCondition` is set to a *don't care value* if the facts are generated from a concrete execution.

4.2. Information Extraction via Symbolic Execution

Symbolic execution is one of the two primary techniques we have considered to extract information from (parametric) test programs. Indeed, being able to observe all execution states across which symbolic execution transits allows us to extract a large amount of information associated with what methods of the SUT are used or, more in general, what parts of the SUT are exercised.

Our solution for extracting behavioural features is based on three main execution phases: i) test program enumeration; ii) feature extraction; iii) Prolog facts generation, according to the format described in Section 4.1. In the following, we detail the methodological/technical organisation of these phases.

Test Program Enumeration The analysis technique is based on JUnit 4/5 annotations and is controlled by a JSON configuration file. This file enables the declaration of those paths to be scanned to find the compiled test classes. The JSON file's structure and the presentation of the configuration it admits are reported in the appendix [5].

Feature Extraction As the symbolic execution engine, we use the Java Bytecode Symbolic Executor (JBSE) [23]. JBSE is a symbolic Java Virtual Machine that deals with complex heap data structures.

At startup, we load all classes associated with test programs declared in the JSON configuration file, all classes associated with the SUT, and all those additional classes are needed to run the application. These paths will be included in the JBSE classpath, enabling the lazy loading of classes on demand. In this way, JBSE can symbolically run all test programs, as we describe below.

To reduce the time required to perform the symbolic execution and focus only on the test programs, we use a form of *concolic execution* [24]. It is essentially a "mixed" concrete/symbolic execution which we use to quickly reach (in a concrete way) each test program's entry point, which is later executed symbolically. This way, we do not explore parts of the execution irrelevant for extracting similarity information, such as those in charge of setting up the environment for a test program execution (e.g., `@Before` or `@BeforeClass` in JUnit), as well as related to multiple mocking frameworks [25] (e.g., `Mockito`).

As already mentioned, we are interested in extracting general information to support multiple decision strategies when similarity measures are constructed at a later stage. To this end, we inspect all symbolic execution states explored by JBSE, and we focus only on the states associated with the invocation of some (local) method. We keep track of all invoked methods, in all explored branches, in an in-memory data structure.

Prolog facts generation When the symbolic execution is completed, we dump a set of `invokes` Prolog facts to a file on disk. These facts are easily derived from the in-memory representation of the symbolic states of interest.

4.3. Information Extraction via Concrete Execution

To collect behavioural information in a concrete execution, we rely on the *Java Agent* technology for the byte-code inspection and manipulation. Specifically, we developed a Java Agent in Javassist, directly attached to the JUnit run, relying on the Maven Surefire plugin. The very first time a class is loaded in memory, we check if it should be instrumented (i.e., it belongs to the test program or the SUT). Thus, the Java Agent injects tracing probes in some specific points of interest.

Specifically, the instrumented methods include a combination of activities performed just after its invocation and before it returns to the caller. These activities allow us to build an in-memory representation of the Prolog facts described in Section 4.1 and also consider the specific test programs that originated the call. Further operative information about the implementation is available in the appendix[5].

4.4. Prolog Facts Processing

The information extraction phase, obtained through either symbolic or concrete execution, generates a knowledge base consisting of `invokes` facts that is used to carry out automated logic-based reasoning to determine test program similarity, according to some (user-specified) criteria.

In order to analyse the sequence of methods executed by running a test program, that is, an *execution trace* of a test program, we provide the predicate `trace(TP,Trace)` (see Listing 7), which relates a test program `TP` to an execution trace `Trace` of the method annotated by `@Test` in `TP`, that is, the entry point of `TP`. The execution trace `Trace` is a list of `invokes` facts whose head `Ep` is the entry point of `TP`.

```
trace(TP,Trace) :-
   tp_entry_point(TP,Ep),
  trace_starting_with(Ep,Trace).
```

Listing 7: Prolog rule that defines `trace(TP,Trace)`.

Now, we can get the execution traces generated by executing a test program by collecting the answers to the query `trace(TP,Trace)`, where `TP` is bound to the test program name and `Trace` is an unbound variable, thereby getting values for `Trace` that can be further processed by using suitable helper predicates, and finally analysed to discover similarity relations between test programs. In particular, we provide the helper predicate `filter`, whose implementation details are reported in the appendix[5], which allows us to sieve through the `invokes` facts and reshape them into suitable data structures to be used within queries for reasoning about the test program similarity. Notably, in testing microservices applications, where we are interested only in analysing the similarity between test programs concerning their remote API invocations, the `filter` predicate allows us to perform the following operations: (1) select those `invokes` facts that represent invocations of methods belonging to remote APIs, (2) extract from the selected `invokes` facts specific information related to the remote API invocation, that is, the HTTP method used to perform the request (e.g., get and post) and its URI, and (3) generate new facts, called `endpoint`, with the following structure:

```
endpoint(TestProgram, Caller, HTTPMethod, URI)
```

These facts showcase that the method `Caller` of the test program `TestProgram` invokes the remote API identified by `URI` using the HTTP method `HTTPMethod`.

In the appendix[5], we also report the query to perform operations (1)–(3).

5. Rules for Similarity

We now present the Prolog rules defining *similarity relations* between test programs, and we show how to use them

[5]The appendix has been submitted in a separate file as "Supplementary material for on-line publication only".

to query the knowledge base consisting of `invokes` and `endpoint` facts for inferring the similarity of test programs.

We start by introducing two basic notions defining the similarity between elements of the *domain*, that is, the similarity between `invokes` facts and between `endpoint` facts. The similarity between elements of the domain is evaluated by using the predicate `matching(Dom,O1,O2)` shown in Listing 8, where `Dom` defines the domain of the elements `O1` and `O2` (either `invokes` or `endpoint`) compared according to the definitions introduced as follows. Given two `invokes` facts `I1` and `I2`, we say that they are similar if and only if (c1) `I1` invokes the same method of `I2` (line 4). Given two `endpoint` facts `E1` and `E2`, we say that they are similar if and only if: (c2) they make use of the same HTTP method to invoke a remote API (line 8), and (c3) their URIs match (line 9).

Every occurrence of an anonymous variable '`_`' represents a distinct variable. It is used to denote any argument that is not taken into consideration for establishing the similarity between `invokes` (and between `endpoint`) facts.

```
1  matching(invokes,I1,I2) :-
2    I1 = invokes(_,_,_,_,_,_,_,Callee1,_),
3    I2 = invokes(_,_,_,_,_,_,_,Callee2,_),
4    Callee1 == Callee2.                     % (c1)
5  matching(endpoint,E1,E2) :-
6    E1 = endpoint(_,_,HTTPMethod1,URI1),
7    E2 = endpoint(_,_,HTTPMethod2,URI2),
8    HTTPMethod1 == HTTPMethod2,             % (c2)
9    matching_URIs(URI1,URI2).               % (c3)
```

Listing 8: Prolog rules that define `matching(Dom,O1,O2)`.

Now, building upon the `matching` predicate, we can define the `similar_tp` predicate, which evaluates the similarity between two test programs.

```
  similar_tp(Dom,DomSrc,SimCr,TP1,TP2,WT1,WT2,Score)
```

Listing 9: Prolog rule that defines `similar_tp`.

The predicate in Listing 9 states that the test program `TP1` is similar to `TP2` according to the similarity criterion `SimCr` based on the matching of elements, belonging to the domain `Dom`, generated during the feature extraction phase from the knowledge-base source `DomSrc`. In particular, if we specify the parameter `trace` for `DomSrc`, the elements of `Dom` are generated from the `invokes` facts occurring in execution traces. `WT1` and `WT2` are lists of elements in `Dom` that witness the similarity of `TP1` and `TP2`, and `Score` is a numeric value that quantifies the *degree of similarity* between `TP1` and `TP2`.

Note that the execution of a test program based on symbolic execution may generate several execution traces, for a pair $\langle \texttt{TP1}, \texttt{TP2} \rangle$ of test programs there may be several pairs $\langle \texttt{WT1}, \texttt{WT2} \rangle$ of witnesses, and hence several score values.

We have defined several similarity criteria specified by means of the `SimCr` parameter of the `similar_tp` predicate. When defining the criterion `SimCr`, we will say that "the similarity criterion `SimCr` holds" as a shorthand for "the predicate `similar_tp` with similarity criterion `SimCr` holds".

First, we present the following set-based similarity criteria:

- `nonemptyEqSet` holds if `WT1` and `WT2` are nonempty lists and every element of `WT1` matches an element of `WT2` and vice-versa;

Table 1: Values of `Score` for set-based `SimCr` similarity criteria. $\pi_{\texttt{Dom}}$ is either (i) the function $\pi_{invokes}$ that, for any `invokes` fact i, returns the `Callee` argument of i, or (ii) the function $\pi_{endpoint}$ that, for any `endpoint` fact e, returns the pair $\langle m, re \rangle$ where m is the `HTTPMethod` argument of e and re is the regular expression accepting the `URI` argument of e.

SimCr	Score
`nonemptyEqSet`	1
`nonemptySubSet`	$\frac{\lvert \text{setOf}(\texttt{WT1},\pi_{\texttt{Dom}}) \rvert}{\lvert \text{setOf}(\texttt{WT2},\pi_{\texttt{Dom}}) \rvert}$
`nonemptyIntersection`	$\frac{\lvert \text{setOf}(\texttt{WT1},\pi_{\texttt{Dom}}) \cap \text{setOf}(\texttt{WT2},\pi_{\texttt{Dom}}) \rvert}{\min(\lvert \text{setOf}(\texttt{WT1},\pi_{\texttt{Dom}}) \rvert, \lvert \text{setOf}(\texttt{WT2},\pi_{\texttt{Dom}}) \rvert)}$

- `nonemptySubSet` holds if `WT1` is a nonempty list and every element of `WT1` matches an element of `WT2`;
- `nonemptyIntersection` holds if there exists an element of `WT1` that matches an element of `WT2`.

The value of `Score` is 0 if the similarity criterion does not hold and, otherwise, it is a non-negative value computed as shown in Table 1, where $\text{setOf}(\texttt{L}, \pi_{\texttt{Dom}})$ denotes the set $\{\pi_{\texttt{Dom}}(o) \mid o$ is an element of $\texttt{L}\}$, for any function $\pi_{\texttt{Dom}}$ on the domain `Dom` of the elements of list `L`.

$$\begin{tikzcd}[column sep=small]
\text{t1} \arrow[r] & \text{m1} \arrow[r] & \textbf{m2} \arrow[r] & \text{m3} \arrow[r] & \textbf{m7} \arrow[r] & \text{m3} \arrow[r] & \textbf{m5} \arrow[r] & \textbf{m6} \arrow[r] & \textbf{m7} \arrow[r] & \text{m11} \arrow[r] & \text{m12} \\
\text{t2} \arrow[r] & \textbf{m2} \arrow[r] & \text{m44} \arrow[r] & \text{m51} \arrow[r] & \text{m88} \arrow[r] & \textbf{m5} \arrow[r] & \textbf{m6} \arrow[r] & \textbf{m7} \arrow[r] & \text{m44} \arrow[r] & \text{m14}
\end{tikzcd}$$

Figure 4: An example of concrete execution traces for the test programs t1 and t2.

As an example, let us consider the concrete execution traces of the two test programs t1 and t2 shown in Figure 4. Specifically, both traces record (relevant) methods that have been invoked when executing the corresponding test programs and their relative invocation order. The methods occurring in the tails of lists starting with t1 and t2, respectively, represent the `Callee` arguments of the `invokes` facts. Given that among the methods invoked by t1 there is m1 that is not invoked by t2, and among the methods invoked by t2 there is m44 that is not invoked by t1, the similarity criteria `nonemptyEqSet` and `nonemptySubSet` between t1 and t2 do not hold. Conversely, the similarity criteria `nonemptyIntersection` holds because t1 and t2 have some method invocations in common, specifically they both invoke the methods m2, m7, m5, and m6. Therefore, the degree of similarity between t1 and t2 is:

$$\frac{\lvert\{\text{m2,m7,m5,m6}\}\rvert}{\min(\lvert\{\text{m1,m2,m3,m7,m5,m6,m11,m12}\}\rvert,\lvert\{\text{m2,m44,m51,m88,m5,m6,m7,m14}\}\rvert} = 0.5$$

We have also defined the following sequence-based similarity criteria for a pair of nonempty lists `WT1` and `WT2` of the form $\langle a_1, ..., a_n \rangle$ and $\langle b_1, ..., b_m \rangle$, respectively:

- `nonemptyEqSeq` holds if $n = m$ and, for $i = 1, \ldots, n$, a_i matches b_i.
- `nonemptySubSeq` holds if $m \geq n$ and, by deleting zero or more elements from `WT2`, we get a list `WT3` such that `nonemptyEqSeq` holds for `WT1` and `WT3`;
- `nonemptyCommonSeq` holds if `nonemptyIntersection` holds.

Table 2: Values of Score for sequence-based SimCr similarity criteria. matchingSeq(L1,L2) is the longest non-empty list L3 such that the similarity criteria nonemptySubSeq holds between L1 and L3, and between L2 and L3.

SimCr	Score
nonemptyEqSeq	1
nonemptySubSeq	$\frac{\text{length(WT1)}}{\text{length(WT2)}}$
nonemptyCommonSeq	$\frac{\text{length(matchingSeq(WT1,WT2))}}{\text{min(length(WT1),length(WT2))}}$

Similarly to set-based criteria, the value of Score is 0 if the similarity criterion does not hold; otherwise, it is a non-negative value computed as shown in Table 2.

Let us consider again the execution traces shown in Figure 3a. Similarly to nonemptyEqSet and nonemptySubSet, the similarity criteria nonemptyEqSeq and nonemptySubSeq do not hold due to the presence of methods invoked by t1 that are not invoked by t2, and vice versa. However, by considering the similarity criterion nonemptyCommonSeq, which holds whenever the criterion nonemptyIntersection holds, the degree of similarity between t1 and t2 is:

$$\frac{\text{length}(\langle m5,m6,m7\rangle)}{\min(\text{length}(\langle m1,m2,m3,m7,m3,m5,m6,m7,m11,m12\rangle), \text{length}(\langle m2,m44,m51,m88,m5,m6,m7,m14,m44\rangle))} = 0.33$$

In this section we have introduced various criteria that aim to evaluate the similarity of test programs by taking advantage of the information about the local and remote API methods they invoke. As mentioned in Section 3, by taking advantage of such information collected during the dynamic analysis of test programs, these criteria can contribute to defining flexible policies within a governance framework for regression testing. Notably, they can be used to select test programs useful to exercise the modified microservices component, and therefore to avoid rerunning all available test programs when a small component changes. In the next sections, we present the validation methodology and the experimental evaluation we have performed to study the performance of the proposed criteria in inferring the similarity among test programs that belong to two test suites.

6. Validation Methodology

In the rest of this section, we first describe the research questions (RQs) that guide our validation methodology (see Section 6.1); then, Section 6.2 presents the two case studies we referred to in our study.

6.1. RQs, Strategies, and Methods

The following presents the RQs we set out to answer in this work. For each RQ, we report the strategy we followed in order to provide an answer and the method we planned to conduct the experimental studies.

RQ1: *Can implicit similarities extracted from test programs support decisions in the context of a governance framework for regression testing?* The behavioural features extracted from each test program represent a valuable source of information that can be exploited while making decisions during the regression testing activities. In this work, we propose a technique to analyze the overlaps (if any) that execution traces of two test programs reveal either during either concrete or symbolic executions. In answering RQ1, we aim to show that our technique is indeed capable of inferring similarities and differences between test programs, according to the criteria defined in section 5, thus making this information available for practical use. In particular, when answering RQ3 defined below, we will argue that the similarity information has a very good potential to support test case reduction in microservices applications. However, as already mentioned, the design and implementation of specific techniques for optimising regression testing are beyond the scope of this paper.

RQ2: *How stable are the similarity criteria?* Implicit similarities among test programs are identified using logic reasoning on the key features carved from their execution traces. Given a set of test programs (i.e., *TS*) for a SUT, a similarity criterion can be used to group test programs in clusters according to an agreed minimum degree of similarity (i.e., s_{min}). We consider that a similarity criterion is stable if the clusters it defines are composed of *homogeneous* test programs: any test program taken from a cluster should be a sample that is good enough to represent all the other test programs in that cluster. In other words, given a threshold s_{min}, all the possible subsets of *TS* built from an arbitrary selection of one element per cluster should always provide comparable outcomes.

To answer RQ2, we planned the following strategy for each similarity criterion. First, we build a subset *TS-small* from *TS*. Specifically, we randomly pick a test program *t* from *TS* and add it to *TS-small* only if its similarity score with all the current elements in *TS-small* is always lower than s_{min} (i.e., *t* is different enough from the elements in *TS-small*). All the test programs in *TS* are considered just once. When this first phase ends, the resulting *TS-small* is run against the referenced SUT, and we register the observed coverage. Both phases are repeated multiple times to experiment with different selections of *TS-small* for the same similarity criterion. The analysis of the coverage data and their variance across several repetitions gives arguments to answer RQ2.

RQ3: *Can the similarity criteria impact the decisions about test suite reduction?* Once the stability of the similarity criteria has been addressed, we are interested in investigating if and how much each similarity criterion can contribute to a test suite selection policy. In other words, we are trying to estimate the quality grain of the proposed selection criteria.

In this sense, we use two software coverage metrics as a consolidated and widely used means of estimating fault detection capabilities: higher levels of code coverage correspond to (but do not guarantee) higher confidence in the ability to detect the presence of bugs. Thus, given a software system and two different test suites for it, the difference in the coverage scored by the test suites estimates their relative potential for defect detection.

In general, any test-suite reduction strategy impacts the SUT coverage: fewer test programs to execute can only decrease the coverage metrics. Our notion of quality is concerned with estimating the coverage drop caused by the selection criteria. We want to exclude as many test programs as possible from the execution, but with limited impact on the coverage of the SUT. In the following, we refer to the quality of a similarity criterion as its capability to define proper subsets of *TS* whose cardinality is smaller than the one achieved by a random selection of test programs in *TS* but resulting in the same coverage drop.

Notably, information on code coverage is of limited use if the aim is to detect defects earlier. However, since this work intends to discover potential similarities between test programs in a microservices application, the answer to RQ3 is limited to an analysis of the relative defect detection capabilities of the considered test suites. Furthermore, in the context of regression testing, it can be assumed that the available test suites are quite stable, while the SUT (frequently) evolves over time. Thus, the calculation of similarities between test programs and their analysis can be done once. The resulting results are expected to be valid until some element in the regression test suites changes. For this reason, the answer to RQ3 does not include any study of the cost of computing the similarities between test programs.

The method we plan to support our validation strategy is similar to the approach described for RQ2. Specifically, for each similarity criterion, we select a subset of test programs *TS-small* as described above[6], and for all the resulting *TS-small* we compute their coverage of the referenced SUT. We repeat these phases several times and then calculate the mean coverage value and the mean number of selected test programs per similarity criterion.

In addition, we also build random subsets of *TS*: we have precisely one random subset of *TS* (i.e., *TS-small-rnd*) for each cardinality between 1 (i.e., a selection with the maximum saving in terms of test programs to be executed) and the total number of test programs in *TS* (i.e., there is no selection and thus no saving). For all these *TS-small-rnd*s, we register their coverage of the SUT. We repeat this procedure several times to compute the mean coverage value expected by a defined-size random selection of test programs.

Thus, we finally answer RQ3 by analyzing the mean coverage outcomes led by the similarity criteria and those resulting from the random selections. Specifically, having agreed on an acceptable coverage drop for running the whole *TS*, we compare the number of test programs in *TS-small* and those in *TS-small-rnd*.

6.2. Subjects

In this section, we introduce the subjects on which we conducted our study about test programs' similarity. The choice of benchmarks for our experimental study was guided by several factors. Firstly, we decided to use benchmarks from the research community rather than building ad-hoc synthetic applications in order to avoid bias in the experiments. This made it possible to test our approach using applications built by third parties in a completely independent way of the testability purposes inherent in our proposal. Given the focus on microservices, we concentrated our selection on all applications implemented according to this paradigm. Furthermore, for technological reasons, we considered all and only those applications written in Java. Finally, the selection focused on applications that provided a test suite of non-minimal size, including integration and contract tests that exercised the microservices API. All these guiding factors led us to select two popular open-source applications, designed according to the microservice architectural style, against which we exercised the reference implementation of our proposal.

The first benchmark that we have used is Fullteaching[7], an educational platform based upon OpenVidu, an open-source video conferencing system employing the WebRTC API [26]. It provides a test suite implemented using JUnit 4, including 88 test programs. Among them, 29 tests require contacting remote URIs for integration or end-to-end testing purposes. These are the test programs used for our case study, as they involve invocating URIs using get, post, put, and delete methods. All RESTful requests are managed through the `MockMvc` class by the Spring framework.

The second benchmark used in our study is a medium-size microservices application called TrainTicket[8], which implements a system for railway ticketing. TrainTicket allows users to inquire about the train tickets between two cities on a certain day, reserve tickets for a specific passenger on a specific class/seat, pay for the reservations (and send the related confirmation email) and manage ticket changes.

TrainTicket comprises 43 total microservices, 38 of which are implemented in Java. These 38 microservices ship with a total of 682 test programs implemented using JUnit 4. All the test programs have been used in our empirical evaluation.

7. Empirical Evaluation

In this section, we present the result of our empirical evaluation based on the applications described in Section 6. The reference implementation of our methodology has been embedded in the Hyperion tool, which is released as open-source software (see the section titled: "Replication Package and Data Availability"). We use the results to answer the RQs listed in Section 6.1.

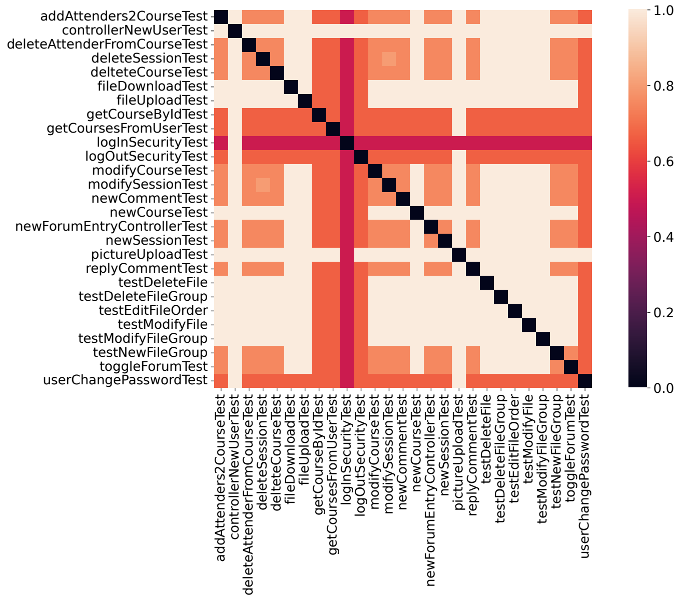
7.1. Capability to Detect Similarity

By the rules discussed in Section 4.4, we have generated the `endpoint` facts that describe the URI(s) invoked by the test programs for both benchmark applications. An excerpt for the Fullteaching application is provided in Figure 6, where we

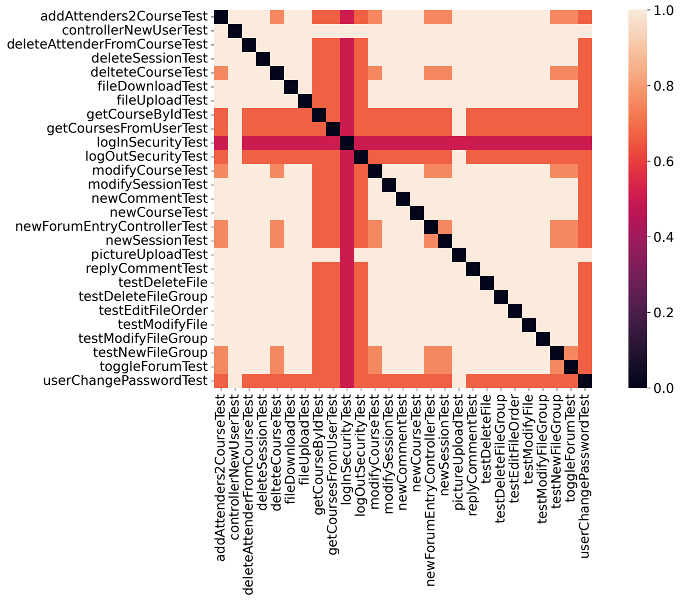
[6]In the empirical evaluation, we have used the same subsets *TS-small* used for RQ2.

[7]`https://github.com/OpenVidu/full-teaching`

[8]`https://github.com/FudanSELab/train-ticket`.



(a) Minimum Similarity Score.



(b) Maximum Similarity Score.

Figure 5: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonempty-Intersection`.

show a subset of the `endpoint` facts generated from the symbolic execution of the two test programs `modifySessionTest` and `deleteSessionTest`. These facts allow us to answer multiple queries, such as: "*which test programs invoke the* `/api-users/new` *endpoint?*", or "*which test programs use the* `/api-courses/new` *RESTful API after* `/api-users/new`*?*". In general, these facts prominently capture that a given test program (the first argument) invokes a certain URI (the fourth argument) with a given method (the third argument), which is fundamental for detecting similarity in the context of microservices applications.

To answer RQ1, we now consider the result of the similarity analysis using various criteria. We consider the results related

```
endpoint('SessionControllerTest:modifySessionTest', '
    registerUserIfNotExists', 'post', '/api-users/new').
endpoint('SessionControllerTest:modifySessionTest', '
    createCourseIfNotExist', 'post', '/api-courses/new')
    .
endpoint('SessionControllerTest:modifySessionTest', '
    newSession', 'post', '/api-sessions/course/1').
endpoint('SessionControllerTest:modifySessionTest', '
    modifySessionTest', 'put', '/api-sessions/edit').
endpoint('SessionControllerTest:deleteSessionTest', '
    registerUserIfNotExists', 'post', '/api-users/new').
endpoint('SessionControllerTest:deleteSessionTest', '
    logIn', 'get', '/api-logIn').
endpoint('SessionControllerTest:deleteSessionTest', '
    createCourseIfNotExist', 'post', '/api-courses/new')
    .
endpoint('SessionControllerTest:deleteSessionTest', '
    newSession', 'post', '/api-sessions/course/1').
```

Figure 6: Example of generated `endpoint` facts. *Subject*: Fullteaching.

to both method invocation and endpoint activation. In the case of TrainTicket, we are considering the complete test suite. We only discuss some exemplary results related to symbolic execution at this point because they enable us to consider a broader range of similarity values and explore additional analysis possibilities. The reader can find the results associated with all the combinations of metrics, domains, and carving techniques in the appendix[5]of this article.

We present similarity results in the form of matrices (heat maps). The value of the similarity score is represented by a coloured cell for each test program pair. Regarding the Fullteaching application, in Figure 5 we present the results related to the `nonemptyIntersection` metric evaluated over the `endpoint` domain. Since symbolic execution can extract multiple traces from the execution of a single test program, no single similarity score value can be associated with a pair of test programs. Therefore, we report in Figure 5a and Figure 5b the minimum and the maximum score values, respectively—the diagonal is zero in all cells, as we do not compute the similarity between a test program and itself. By construction, the similarity matrix for the `nonemptyIntersection` metric is symmetric.

To understand whether this information can be used effectively to detect implicit similarities between test programs in the context of a governance framework for regression testing, let us discuss some values related to Figure 5a and Figure 5b. If we consider the test program `logInSecurityTest`, which tests the login capabilities of Fullteaching users, we observe that it is associated with a minimum/maximum similarity score of 0.5 with respect to all other test programs. The test programs we have taken into account are all associated with authenticated APIs: all test programs try to create a user (if it does not exist), authenticate it, perform some action, and conclude the session. Therefore, `logInSecurityTest`'s similarity score is stable compared to the other test programs, and it is set to a low value. In this sense, we cannot consider it significantly similar to other test programs.

Let us now focus on the test program `deleteSession-Test`. If we compare the minimum and maximum scores against `newCourseTest` and `getCourseByIdTest`, we may try to an-

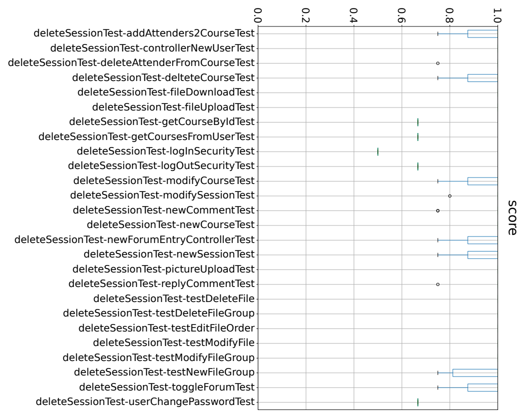
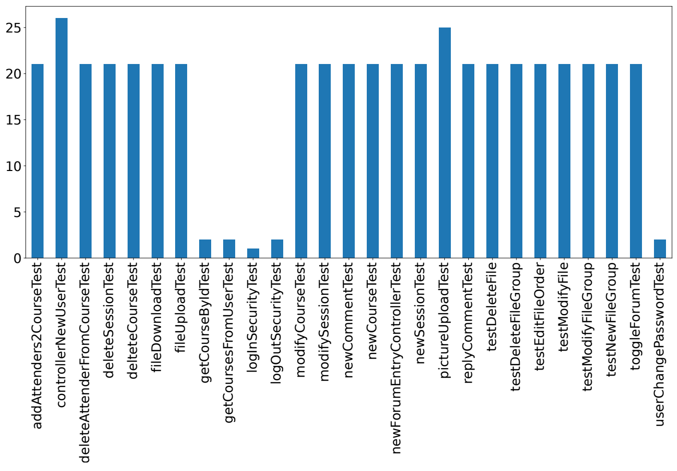


Figure 7: Effect of Multiple Symbolic Traces on Pairs of Test Programs. *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyIntersection`.

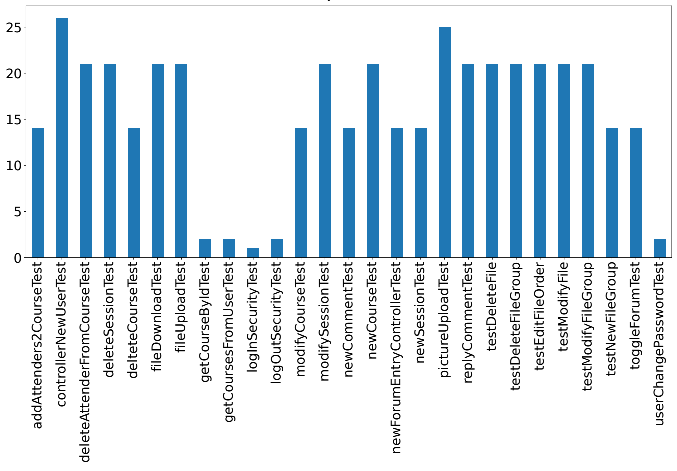
swer the question: "*to which test is* `deleteSessionTest` *most similar?*". The pair `deleteSessionTest–newCourseTest` is associated with a minimum/maximum value of 1.0, while `deleteSessionTest–getCourseByIdTest` has a minimum/maximum value of 0.75. We might conclude that, as far as endpoint invocations are concerned, `deleteSessionTest` is more similar to `newCourseTest` than `getCourseByIdTest`. On the other hand, if we compare the values of the pairs `deleteSessionTest–modifySessionTest` and `deleteSessionTest–getCourseByIdTest`, the pair `deleteSessionTest–modifySessionTest` shows a minimum value of 0.75 and a maximum value of 1.0 (depending again on the multiple observed symbolic execution traces), while `deleteSessionTest–getCourseByIdTest` is stable at 0.67. In this case, we cannot conclude much on the similarity among `deleteSessionTest`, `modifySessionTest`, and `getCourseByIdTest`.

However, if we observe the results in Figure 7, we can extract more information. In the figure, we have picked `deleteSessionTest` and displayed the dispersion of the similarity score compared to all the other test programs. By looking at these results, we might conclude that `deleteSessionTest` is more similar to `modifySessionTest` than `getCourseByIdTest`. Conversely, let's also consider `newSessionTest`. We might conclude that `deleteSessionTest` is more similar to `newSessionTest` than `getCourseByIdTest`, but not as much as we might imagine by looking at Figure 5. It is also interesting to note that, for some pairs (e.g., `deleteSessionTest–loginSecurityTest`), there is no dispersion at all—this is also reflected in Figure 5, where both the minimum and maximum values are the same. This phenomenon can be related to the fact that, in the symbolic execution tree, there is only one feasible path for the test program `loginSecurityTest`. In contrast, for other test programs, there are multiple execution traces to compare; therefore, different similarity scores are derived.

In Figure 8, we show the number of test programs that can be deemed similar by relying on our metric. In particular, for each test program, we report the number of other test programs that have a median similarity score among all symbolic execution traces above 0.75 (Figure 8a) and exactly 1.00 (Figure 8b). As expected, the number of test programs deemed similar decreases for higher median values. This result is an additional indication of the versatility of our approach. Indeed, higher similarity score values might help define narrower governance policies that can be enforced to reduce regression test suites by selecting some test cases, skipping redundant ones, or prioritizing those expected to yield earlier fault detection.



(a) Similarity Score ≥ 0.75.

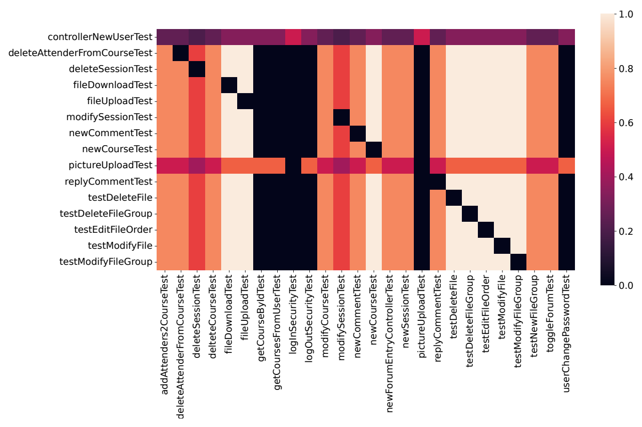


(b) Similarity Score = 1.00.

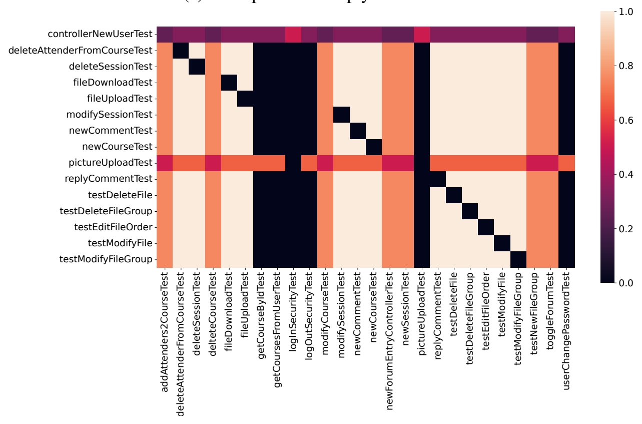
Figure 8: Number of Test Programs Deemed Similar. *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyIntersection`.

In Figure 9 we report the similarity matrices (again, distinguishing between the minimum and maximum score values) for the `nonemptySubSet` criterion. As expected from the definition of `nonemptySubSet`, we observe from the results that this criterion provides non-symmetric results. The first important difference compared to the results in Figure 5 is the different cardinality of the sets of test programs deemed similar. In particular, more conservative similarity criteria, such as `nonemptySubSet`, consider as similar fewer test programs than more inclusive criteria such as the aforementioned `nonemptyIntersection`.

The values of the similarity scores obtained by the different criteria are also interesting to discuss. The `nonemptyEqSet` criterion (see Figure 10) associates each pair of similar test programs with the value 1—non-similar test programs are

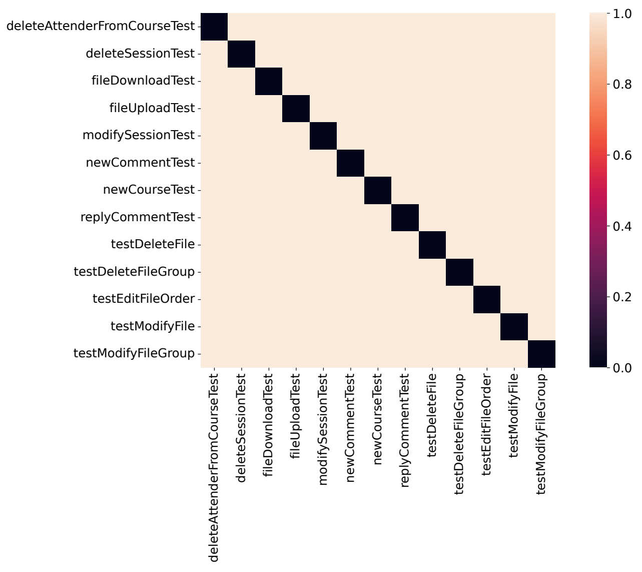


(a) ft:endpoint-nonemptySubSet-min

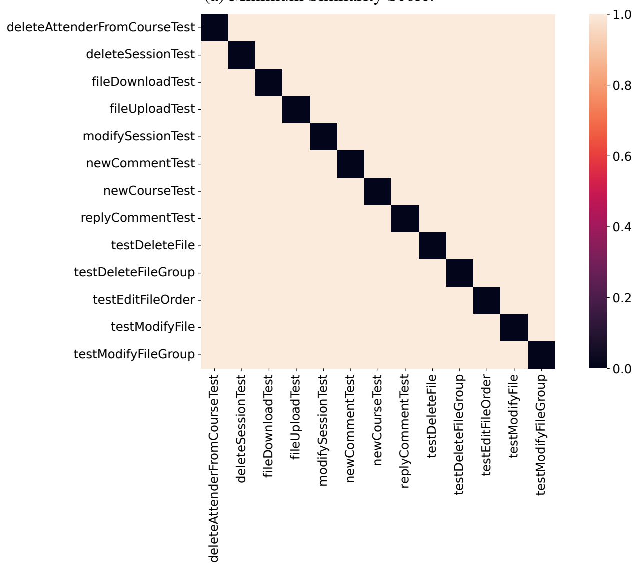


(b) ft:endpoint-nonemptySubSet-max

Figure 9: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptySubSet`.



(a) Minimum Similarity Score.

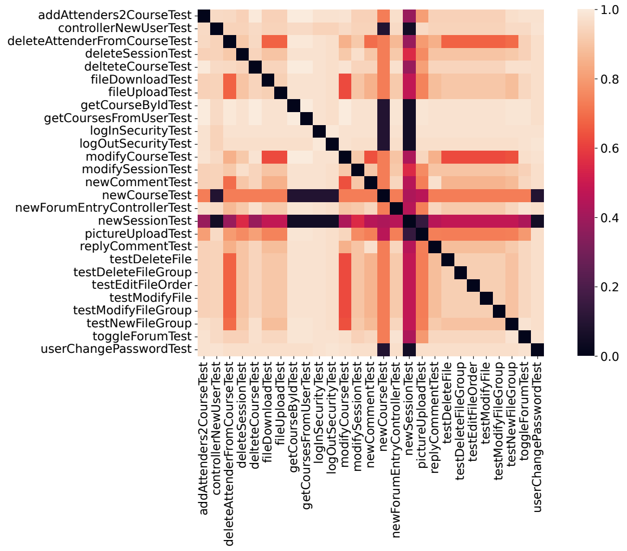


(b) ft:invokes-nonemptyEqSeq-max

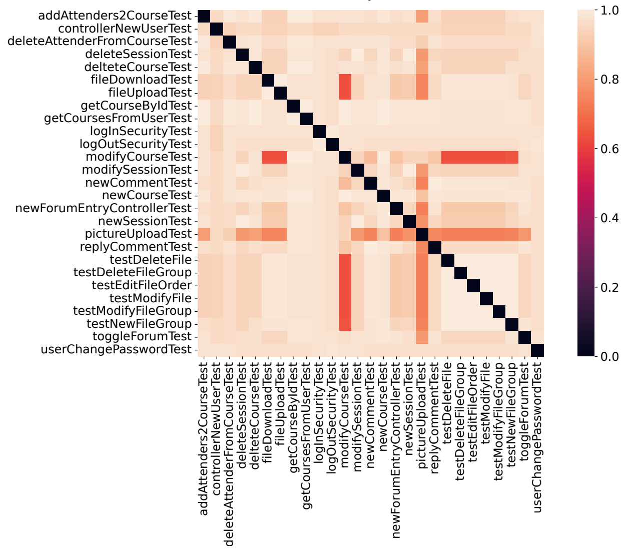
Figure 10: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptyEqSeq`.

not shown. Therefore, this criterion behaves very selectively, deeming two test programs either as (fully) similar or not. This criterion is even more selective than `nonemptySubSet` (fewer test programs are deemed similar). Yet, it is more difficult to discriminate the relative similarity between pairs of test programs due to the boolean nature of the similarity score. Conversely, the aforementioned `nonemptySubSet` criterion shows a (small) number of intermediate similarity score values, while slightly increasing the number of test programs deemed similar compared to `nonemptyEqSet`. If we compare the results in Figures 9 and 10, we notice that many pairs have been evaluated as similar also by the `nonemptyEqSet` similarity criterion with the same score. Indeed, this is expected by the definition of `nonemptyEqSet`, as every time that `nonemptyEqSet` assigns a score 1, so does `nonemptySubSet`. Nevertheless, `nonemptySubSet` is slightly more inclusive, and captures also the fact that some test programs are "not completely" similar, a notion that could be fruitfully exploited when prioritizing the execution of test programs.

Concerning the similarity comparison based on `invokes`, we only present here the results related to the `nonemptyIntersection` and `nonemptyCommonSeq`—the complete exper-



(a) Minimum Similarity Score.



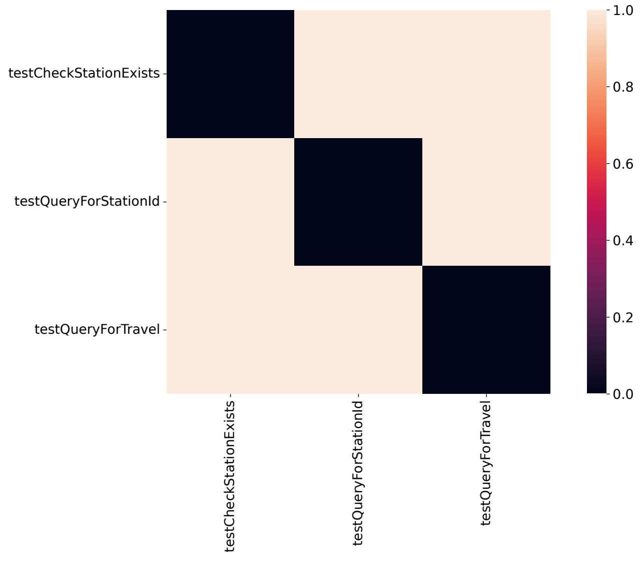
(b) Maximum Similarity Score.

Figure 11: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonempty-Intersection`.

imental data are again located in the appendix[5]. An interesting result can be observed by comparing Figures 5 and 11. Indeed, the results are mostly comparable. This result is related to the nature of the test suite in Fullteaching. Indeed, the test programs that contact some remote endpoints also directly exercise non-minimal parts of the SUT as if they were compounded unit tests. If a test program contacts the same endpoint, it will likely exercise the same parts of the SUT. This behaviour is not common for all test suites. Indeed, in Figure 12 we report the results for both invokes and endpoints in the case of TrainTicket, for the same `nonemptyCommonSeq` criterion—we only report the minimum scores. As can be seen, the results are highly different. The TrainTicket test suite is such that few test programs exercise the same endpoints. Conversely, the SUT is directly exercised more at large. This characteristic is clearly emerging from the results, considering the relevantly different number of test programs deemed similar by the same metric using the two different domains and the more diversified similarity scores observed in the `invokes` case.

To conclude the analysis, in Figure 13 we present the results related to Fullteaching when using the `nonemptyCommonSeq` criterion. When compared to `nonemptyIntersection` (Figure 11), we observe that the number of test programs considered similar is the same. Nevertheless, the score values are more scattered in the range. By the definition of the criteria, this is an expected result. Indeed, considering sequences rather than sets allows us to gather more stringent similarity information. An analysis based on `nonemptyCommonSeq` (and based on sequences in general) could enable a more fine-tuned selection of test programs in the considered governance framework for regression testing.

Overall, the criteria provide results that are comparably different. `nonemptyEqSet` is a stronger similarity criterion, which anyhow leaves out many test programs from the suite. `nonemptyIntersection`, on the other hand, includes a larger number of test programs while being less "categorical" about the similarity between test programs. `nonemptySubSet` and `nonemptyCommonSeq` capture capabilities of both criteria. Concerning RQ1, we can conclude that the criteria can detect similarities between test programs to various degrees, which can be beneficial depending on the current phase of the application's lifecycle. For example, when dealing with testing during feature development, the `nonemptyEqSet` criterion might help determine what test programs to execute after a failure to reduce the time to completion of the test suite—a test program similar to the failed one might be skipped. Conversely, the `nonemptyIntersection` criterion might help in determining what test programs could be run in parallel before releasing a new stable version of the application, e.g., in the possible attempt to detect reentrance bugs—multiple test programs that invoke methods from the same package of the application might be run concurrently.



(a) *Domain*: `endpoint`.

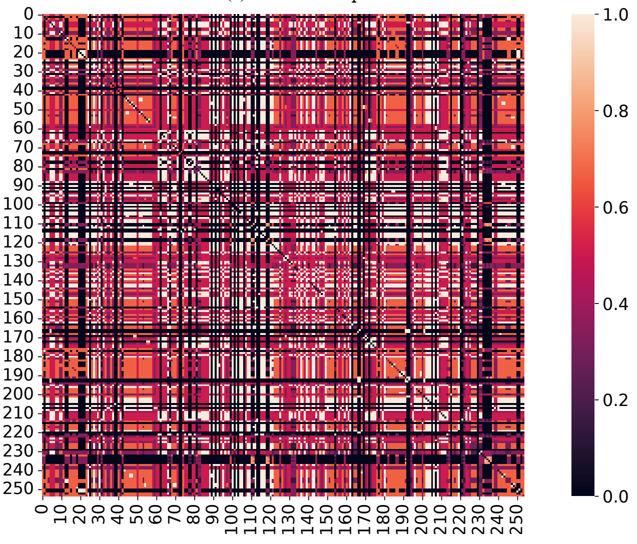
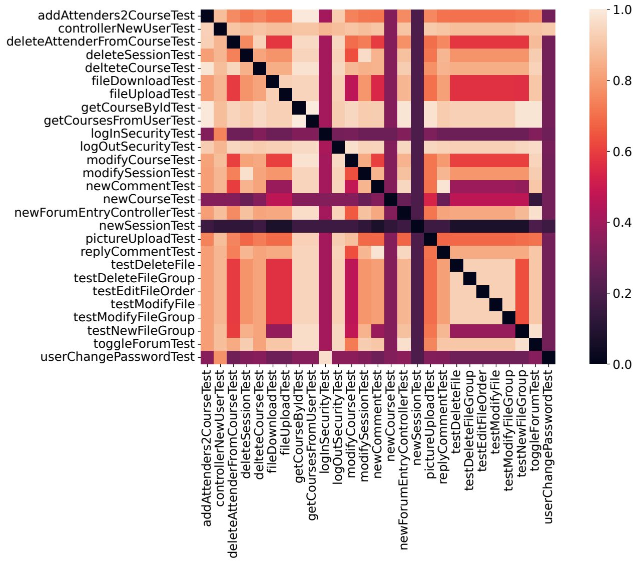

(b) *Domain*: `invokes`.

Figure 12: *Subject*: TrainTicket; *Criterion*: `nonemptyIntersection`.


(a) Minimum Similarity Score.

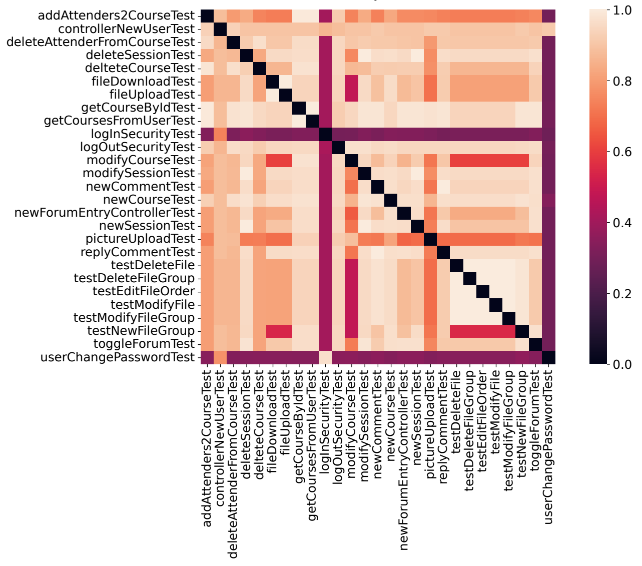
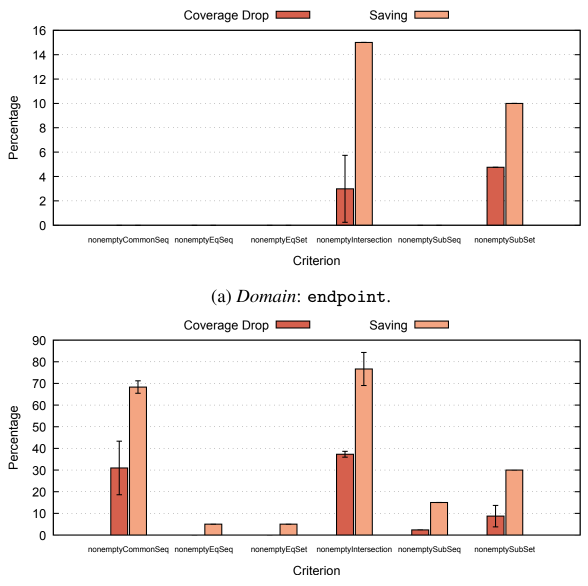

(b) Maximum Similarity Score.

Figure 13: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonempty-CommonSeq`.

Answer to RQ1: The proposed criteria can detect implicit similarities between test programs to various degrees: some criteria are more inclusive and they highlight coarse-grained implicit similarities, while others are more conservative as they report only very narrow similarities. Overall, the implicit similarities extracted from test programs can be used to support decisions in the context of a governance framework (as also argued when answering RQ3 below). However, the actual benefit they provide likely depends on the current phase of the application's lifecycle and needs the development of specific techniques, whose design is beyond the scope of the present paper.

7.2. Stability of the Similarity Criteria

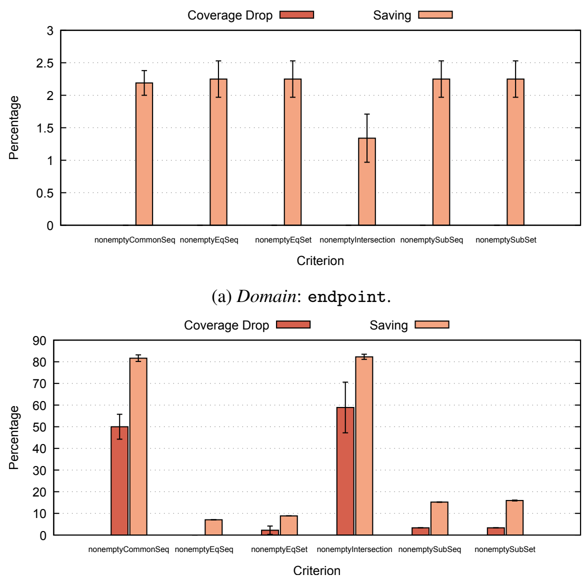
In this part of our empirical evaluation, we explicitly pursue an answer to RQ2. We have not considered symbolic execution traces for this part of the analysis precisely due to their symbolic nature. Indeed, we focus only on traces generated by instrumented execution because they provide a single score for each test program. This approach is helpful when studying the stability of the criteria because it removes a possible source of instability related to the multiple paths explored by the symbolic execution rather than to the criteria themselves. Indeed, symbolic traces would explore execution paths that might not be taken by the actual execution of the test program. Moreover, we target a comparison with a randomly-selected test suite built without considering the symbolic trace. Building *TS-small* based on symbolic execution would lead to incomparable re-



(a) *Domain*: `endpoint`.

(b) *Domain*: `invokes`.

Figure 14: *Subject*: Fullteaching.



(a) *Domain*: `endpoint`.

(b) *Domain*: `invokes`.

Figure 15: *Subject*: TrainTicket.

sults.

As mentioned in Section 6.1, we focus on multiple *TS-small* subsets of the original test suite and study the coverage variance to consider a similarity criterion as stable. We have set the similarity threshold s_{min} to 0.5 for both applications as an intermediate value, allowing a non-minimal number of test programs to be deemed similar. Our experimental assessment has shown that if the threshold is changed, the trends in the experimental results are comparable, although with different slopes related to the increased/reduced number of test programs included. Coverage data have been obtained by relying on JaCoCo [27]. We report the result of this experiment when considering three different *TS-small* subsets for each configuration, in Figures 14 (for Fullteaching) and 15 (for TrainTicket).

We observe from the results that the coverage drop is pretty stable in all configurations as far as the `endpoint` domain is concerned. The only exception is in the Fullteaching case when relying on the `nonemptyIntersection` criterion. As discussed above, `nonemptyIntersection` is the less stringent criterion. Therefore, the *TS-small* suite built based on this criterion is likely to offer a more significant number of selection possibilities. The high variability is also related to many test programs overlapping in the Fullteaching test suite concerning the invoked endpoints. Therefore, a totally-random selection with a larger degree of freedom can produce the observed high variability.

Interestingly, many criteria provide a 0% coverage drop in Fullteaching. This result is an early indication of the capability of the proposed criteria to support the exclusion of test programs that are likely to exercise the same parts of the SUT. This behaviour is more evident in the TrainTicket case (see Figure 15).

Concerning the `invokes` domain, we observe more variability. Besides the already mentioned `nonemptyIntersection` (particularly in TrainTicket), also `nonemptyCommonSeq` shows high variability. The reason for this variability can be found in Figure 16, where we report the associated heatmap. Indeed, from the results, we note that the largest part of the test programs is considered similar, with scores that are around 0.5. Therefore, also in this case, given the large number of test programs and similar values for the scores, the degree of freedom is quite high.

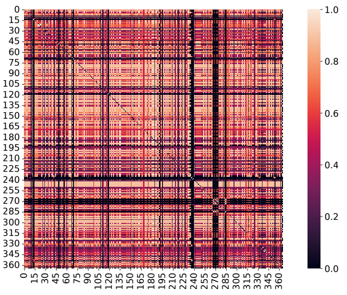
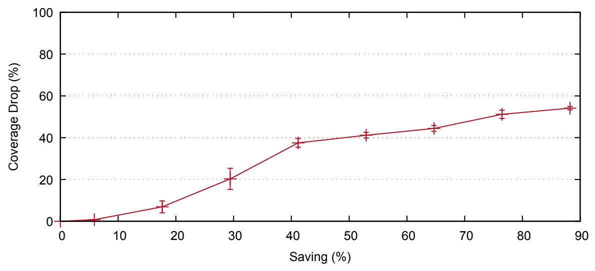


Figure 16: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonempty-CommonSeq`.

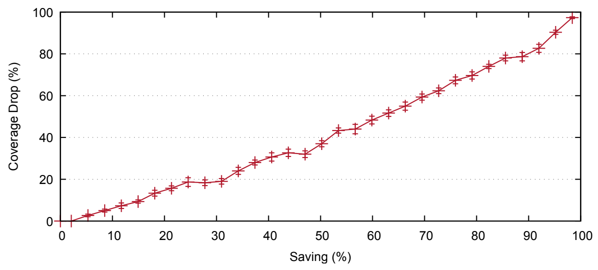
Answer to RQ2: The most stable criteria are the most stringent ones (i.e., `nonemptyEqSet` and `nonemptyEqSeq`). For more inclusive criteria, the stability significantly depends on the nature of the test suite.

7.3. *Quality of the Similarity Criteria*

To answer RQ3, we consider the results provided in Figure 17, where we report the average coverage drop obtained when a certain percentage of test programs is dropped. These results have been collected by running ten different randomly-constructed test suites (*TS-small-rnd*) for each point in the plots—we show average values and 90% confidence intervals. These



(a) *Subject*: Fullteaching.



(b) *Subject*: TrainTicket.

Figure 17: Results with *TS-random*.

data allow answering the following question: "*if you plan to skip x% test programs to save the time to run the test suite, what is the coverage drop that you must be prepared to observe if you have no way to make an informed selection of the test programs?*". As an example, in Fullteaching (Figure 17a), if you skip 60% of the test programs, on average, you can observe a 40% performance drop. Similarly, if you skip 60% of test programs in TrainTicket (Figure 17b), you should be prepared to face a 50% coverage drop.

Therefore, to answer RQ3, we want to determine whether, relying on the similarity criteria that we have proposed, it is possible to obtain a certain level of test program execution saving associated with an equal or lower coverage drop. In Figures 14 and 15, we also present the test program savings we have observed when running the various *TS-small* test suites generated based on the similarity scores. By comparing the results in Figures 14, 15, and 17, we notice that no domain/criterion configuration shows a coverage drop higher than in the case of a random selection of test programs, with the same amount of saving. Most notably, many configurations exhibit a lower coverage drop.

Answer to RQ3: Without any prior knowledge of the test suite, the proposed similarity criteria can extract similarity information from the test programs in a way that can be effectively exploited in governance frameworks for regression testing, and specifically for test suite reduction.

8. Threats to Validity

In the following, we discuss some of the threats that may potentially affect our empirical evaluation's validity, and thus the validity of the conclusions we have drawn.

8.1. Threats to Construct Validity

Any explicit or implicit assumption concerning the setup of the validation scenarios may lead to questionable conclusions. In the following, we discuss the more relevant decisions that may have impacted the interpretation of the observed outcomes.

Knowledge Base Carved from the Test Programs: Implicit dependencies across test programs are extracted by considering a well-defined but limited set of statements coded within the tests' implementations. Indeed, our approach considers only those statements that activate shared local/remote APIs. While presenting our work's research context and motivation, we discussed why these information sources could help study test programs' similarities in microservices applications. However, we agree that false-positive and false-negative similarities may result from this narrowed analysis of the test programs' traces. As we also acknowledged in Section 7.2 and Section 7.3, these considerations become much more relevant when employing the symbolic execution traces. Thus we cannot exclude that a more sophisticated analysis of the behaviour exposed by test programs' implementation may lead to a finer observation of actual similarities.

Inference Rules: Similarly to the specific constructs in the test programs, the way we processed the information collected during the concrete/symbolic executions may concern the appropriateness of the analysis. Section 5 reports the set of inference rules which define the considered similarity criteria. In other words, these inference rules represent the core definitions of test programs' implicit dependencies. Clearly, a different set of similarity criteria could lead to different results and conclusions, but we can argue that the study is limited only to those criteria. However, even focusing on the given group of similarity criteria, we are aware that the current definitions of the rules in Prolog may suffer from typical implementation issues (e.g., related to the backtracking strategies). Such issues may lead to tiny differences between the outcome expected by the abstract formulation of the similarity criterion and the outcome returned by its implementation in a Prolog rule. The Prolog implementations referred to in this work are built, extended, and refined on a set of inference rules from previous works [16] [17]. In addition, before their exploitation in our empirical evaluation, we carefully analyzed them in peer-reviewing sessions that also included dedicated testing activities that should have mitigated such risks.

8.2. Threats to Internal Validity

This class of threats to validity refers to those aspects that could have influenced the observed outcome.

Choice of the Case Study: The empirical evaluation referred to two different subjects. They have been selected because both projects are open-source, abiding by the microservices architectural style, and their development toolchain could be integrated easily into our reference implementation. Nevertheless, both subjects may have hidden or uncontrolled influences on the experimentation, and more case studies may be useful for a more thorough experimental evaluation of our approach.

Minimum Degree of Similarity: The validation methods adopted for answering RQ2 and RQ3 rely on the parameter s_{min} to cluster test programs for a given similarity criterion (see Section 6.1). In these specific settings, we considered two test programs similar if their score was higher than 0.5 over a range $[0-1]$. We are aware that a different tuning of this parameter impacts the presentation of both the quality and the stability of the similarity criteria. However, the results for both concepts are also influenced by the specific composition of the available test suites. Thus, we tried to make a neutral choice that could also overcome the quality/variety of the actual test suites in the considered subjects.

8.3. Threats to External Validity

The expected scenario for a study is to draw conclusions to such an extent that they are valid also in other studies. However, generalised conclusions are difficult to achieve as various factors often threaten them.

Statistical power: Overall the study considered a set of 711 test programs: 29 from Fullteaching, and 682 from TrainTicket. Though the number is not small, it is also evident that it is insufficient to advocate a strong significance for the observed outcomes. In addition, the experimental evaluation only focused on two subjects. Thus our interpretations could be influenced by hidden aspects present in both subjects.

Generalisation: The results we collect in this study may strictly depend on the experiments we planned for the specific case studies we selected. Thus, we cannot draw fully general conclusions that claim the proposed approach can always provide valuable results for any application. We clarify that such a statement needs more extensive validation against different case studies.

In order to support other researchers to repeat our experience or replicate it with different subjects, we make available the whole artefacts developed within the context of this work (see the section titled: "Replication Package and Data Availability"). Also, the appendix[5]details all the similarities of the test programs we produced during this study. We believe this information could support future works aiming to validate the generalisation of the outcomes observed in this study.

9. Related Work

Regression testing is an interesting application context that has been intensively investigated, as reported, for instance, in the survey paper by Yoo *et al.* [5]. Some of the works classified in the survey focus on discovering and processing test cases in a given test suite that *traverse* modifications in the original SUT. Among others, the survey reports on approaches that leverage analysis on control [28, 29] or data [30] flows, symbolic execution [31], textual difference in source code of the SUT [32]. Our work complements these approaches by starting the analysis of similarity from the test program implementations. The results reveal additional information that can potentially be used in other common regression testing activities such as test case prioritisation, minimisation or selection.

A recent systematic mapping study by Waseem *et al.* [33] surveys techniques for testing microservice-based applications. Among these, the paper by Sotomayor *et al.* [34] targets open-source testing tools for microservises.

Some of the surveyed techniques propose the use of formal methods (e.g., model checking) for automated testing, specifically for test case generation, scheduling, and execution [35, 36, 37, 38].

In order to improve the effectiveness of testing, Rahman *et al.* [39] introduce a framework for parallel execution of tests that, by cutting down the execution time, enables frequent re-running of the entire test suite during the development of microservices-based applications. While we share the same goal, and indeed our framework can contribute to the design of flexible policies for effective and efficient regression testing, our approach is closer to those aimed at avoiding re-running all available tests when a small component of the SUT changes, and instead supporting the selection of tests that are useful to exercise the changed component. In order to retrieve subsets of test cases required to deal with microservice changes, Ma *et al.* [40] propose a graph-based approach for analyzing the dependencies among microservices based on their APIs. A dynamic analysis of microservices, based on their workload characteristics, is used by Schulz *et al.* [41] to generate tailored test cases for exercising specific microservices of an application in isolation.

However, the issue of inferring dependencies and similarities among test programs has received very little attention, especially compared to their structural or behavioural analysis. Indeed, instead of analyzing microservices, our approach is based on the dynamic analysis (either concrete or symbolic) of the test suite to determine which microservices are tested (therefore, allowing the selection of only those required to test the modified component).

In the more general context of automated software engineering, the problem of identifying similarities among test programs is related to the broader issue of identifying similarities among generic programs [42], which has been studied for multiple purposes (and with different techniques), such as duplicate code detection [43], plagiarism detection or copyright infringement [44], and code compression [45].

The idea of using test case similarity to design effective testing strategies has been explored in several works [46, 47, 48, 49, 50]. Noor and Hemmati [46] propose an approach for prioritizing test cases based on their similarity with those that failed on previous versions of the software system under consideration. The similarity between test cases is defined by comparing sequences of method calls extracted from execution traces. That paper uses concrete execution while we perform a logic-based similarity analysis on traces extracted via symbolic execution.

Another similarity-based approach for regression test case prioritisation is presented by R. Wang *et al.* [47]. The execution order of the test cases is scheduled based on the distance between them, where the notion of distance is defined concerning branch coverage. That paper evaluates six similarity measures and shows that Euclidean distance gives the best result through experiments on a few benchmark programs.

Test case similarity is defined by Ledru *et al.* [49] based on the string distance between the test cases, and hence no notion of execution is considered. Also Miranda *et al.* [50] base their test case prioritisation technique on similarity relations defined on test cases and not on their execution. To enforce scalability, the similarity is computed by algorithms usually applied in the context of big data processing.

Similarity has been exploited for fault-localisation in the paper by Hao *et al.* [48], where the similarity between test cases is defined by using a fuzzy set representation of a matrix relating test cases and program statements, and candidate faulty statements are selected on a probabilistic basis.

Some papers propose techniques for computing the similarity of programs (not necessarily test programs) based on static analysis or fuzz testing, whereas we employ symbolic execution. In particular, Raman *et al.* [51] use the call-dependency relation among program APIs to generate a trace of the API calling sequence. S. Wang and Wu [52] present a method that uses fuzz testing for similarity analysis of binary code. The similarity score of two behaviour traces generated by fuzzing from two program functions is computed according to their longest common subsequence.

In automated software testing, symbolic execution has been largely used as an effective technique for finding errors in software applications and generating high-coverage test suites [53, 54, 55, 56, 24, 14, 57, 58]. This technique, which was first introduced in the mid 1970's, has been conceived to exercise a software system by searching for potential configurations/states violating a given set of assertions. The basic idea of symbolic execution is strongly related to techniques for *bounded model checking* of software, which use SMT solvers for checking that a specified program property is not violated by any execution path up to a given length bound [59].

The symbolic exploration of the software of states requires the generation of a very complex combination of constraints. The resolution of these constraints frequently leads pure symbolic approaches to suffer severe scalability issues. *Concolic* approaches mitigate such a risk by combining symbolic evaluation with concrete execution and, in some cases, random data generation [54, 60, 55, 24, 56].

In implementing our technique, we use the JBSE, a symbolic Java Virtual Machine which can deal with complex heap data structures. We also use a form of concolic execution to handle methods in charge of setting up the environment for a test program execution (e.g., `@Before` in JUnit). However, the main goal of our work is neither the search for errors nor the generation of test cases. In fact, we want to infer relations between test programs, e.g., various forms of dependency or similarity, and we do so by extracting high-level information from symbolic execution paths and states. Our approach is particularly suitable when dealing with parametric test programs.

Some techniques for *relational verification* make use of *constraint logic programming* (i.e., logic programming augmented with constraint solving) to verify relations between programs [61, 62]. However, the kind of properties targeted by relational verification is very strong (in general, undecidable) relations, such as full functional equivalence, while here we focus on test programs, and we are interested in much weaker dependency and similarity relations based on suitable abstractions of the finite set of paths generated by symbolic execution. In this respect, our work parallels symbolic execution techniques for crosschecking optimized versions of data-parallel programs against the unoptimized ones [63].

10. Conclusions and Future Work

We have discussed a methodology to extract similarity relations among test programs for microservices applications. By dynamic analysis (i.e., either instrumented or symbolic), we can extract from a test suite relevant information about the methods called by the test programs. A set of Prolog rules processes this information to filter the execution traces of interest and generate additional facts to enlarge the knowledge base, e.g., to determine the endpoints that may be activated by running the various test programs. Other Prolog rules compute a similarity score according to multiple criteria. In two case studies, we have observed that our approach can generate a significant amount of information, which can be used in the context of a governance framework for regression testing, for example, by supporting decisions that could prevent the enforcement of a *retest-all* strategy.

Additionally, our empirical evaluation shows that the proposed criteria support the selection of test programs that can be stable and effective at automatically identifying what test programs shall be excluded. Nevertheless, the overall quality of the test suite offered by the application being tested plays a significant role in the selection power of the proposed approach.

Future work includes devising additional rules to cope with test suites that have a high overlap degree across different test programs. Moreover, we plan to exploit the similarity relations to support an online selection procedure that could quickly determine what test programs to execute after the outcome of a previous set of executed test programs is gathered.

Replication Package and Data Availability

The source code used in the empirical evaluation of this article is available at `https://github.com/IASI-SAKS/hyperion/releases/tag/journal`. We also provide the replication package of the experiments in the repository.

Acknowledgments

This paper has been supported by the Italian MIUR PRIN 2017 Project: SISMA (Contract 201752ENYB), and partially by the Italian Research Group: INdAM-GNCS.

References

[1] T. Cerny, M. J. Donahoo, M. Trnka, Contextual understanding of microservice architecture: current and future directions, ACM SIGAPP Applied Computing Review 17 (4) (2018) 29–45.

[2] J. Lewis, M. Fowler, Microservices, a definition of this new architectural term (Mar. 2014).
URL `https://martinfowler.com/articles/microservices.html`

[3] A. Bertolino, G. De Angelis, F. Lonetti, Governing regression testing in systems of systems, in: Proc. of ISSRE Workshops, GAUSS, IEEE, 2019, pp. 144–148. `doi:10.1109/ISSREW.2019.00064`.

[4] T. Clemson, Testing strategies in a microservice architecture (Nov. 2014).
URL `https://martinfowler.com/articles/microservice-testing/`

[5] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability 22 (2) (2012) 67–120.

[6] A. Vahabzadeh, A. Stocco, A. Mesbah, Fine-grained test minimization, in: Proc. of ICSE, 2018, pp. 210–221.

[7] R. Kazmi, D. N. A. Jawawi, R. Mohamad, I. Ghani, Effective regression test case selection: A systematic literature review, ACM Comput. Surv. 50 (2) (2017) 29:1–29:32.

[8] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, R. Tumeng, Test case prioritization approaches in regression testing: A systematic literature review, Information and Software Technology 93 (2018) 74 – 93.

[9] D. Paterson, J. Campos, R. Abreu, G. M. Kapfhammer, G. Fraser, P. McMinn, An empirical study on the use of defect prediction for test case prioritization, in: Proc. of ICST, IEEE, 2019, pp. 346–357.

[10] L. Mariani, S. Papagiannakis, M. Pezzè, Compatibility and regression testing of cots-component-based software, in: Proc. of ICSE, IEEE CS, 2007, pp. 85–95.

[11] M. J. Harrold, A. Orso, Retesting software during development and maintenance, in: Proc. of Frontiers of Software Maintenance, 2008, pp. 99–108.

[12] L. Gazzola, M. Goldstein, L. Mariani, I. Segall, L. Ussi, Automatic ex-vivo regression testing of microservices, in: Proc. of AST, ACM, 2020, pp. 11–20.

[13] M. Bruno, G. Canfora, M. D. Penta, G. Esposito, V. Mazza, Using test cases as contract to ensure service compliance across releases, in: Proc. of ICSOC, Vol. 3826 of LNCS, Springer, 2005, pp. 87–100. `doi:10.1007/11596141\_8`.

[14] C. Cadar, K. Sen, Symbolic execution for software testing: three decades later, Commun. ACM 56 (2) (2013) 82–90.

[15] J. Wielemaker, T. Schrijvers, M. Triska, T. Lager, Swi-prolog, Theory and Practice of Logic Programming 12 (1-2) (2012) 67–96.

[16] E. De Angelis, G. De Angelis, A. Pellegrini, M. Proietti, Inferring relations among test programs in microservices applications, in: Proceedings of the 15th IEEE International Conference on Service Oriented Systems Engineering, SOSE, IEEE, 2021, pp. 114–123. `doi:10.1109/SOSE52839.2021.00018`.

[17] E. De Angelis, A. Pellegrini, M. Proietti, Automatic extraction of behavioral features for test program similarity analysis, in: Proc. of ISSRE Workshops, GAUSS, IEEE, 2021, pp. 129–136. `doi:10.1109/ISSREW53611.2021.00054`.

[18] R. Baldoni, E. Coppa, D. C. D'Elia, C. Demetrescu, I. Finocchi, A survey of symbolic execution techniques, ACM Comput. Surv. 51 (3) (2018) 50:1–50:39.

[19] Q. Yang, J. J. Li, D. M. Weiss, A survey of Coverage-Based testing tools, Computer Journal 52 (5) (2009) 589–597. `doi:10.1093/comjnl/bxm021`.

[20] E. Bruneton, R. Lenglet, T. Coupaye, ASM: a code manipulation tool to implement adaptable systems, Adaptable and extensible component systems 30 (19) (2002).

[21] S. Chiba, Load-Time structural reflection in java, in: E. Bertino (Ed.), ECOOP 2000 — Object-Oriented Programming, Vol. 1850 of Lecture Notes in Computer Science, Springer Intrernational Publishing, Berlin Heidelberg, Germany, 2000, pp. 313–336. `doi:10.1007/3-540-45102-1\_16`.

[22] B. García, F. Lonetti, M. Gallego, B. Miranda, E. Jiménez, G. De Angelis, C. E. Moreira, E. Marchetti, A proposal to orchestrate test cases, in: Proc. of QUATIC, 2018, pp. 38–46.

[23] P. Braione, G. Denaro, M. Pezzè, JBSE: A symbolic executor for Java programs with complex heap inputs, in: Proc. of FSE, ACM, 2016, pp. 1018–1022. `doi:10.1145/2950290.2983940`.

[24] K. Sen, Concolic testing, in: Proc. of ASE, ACM, 2007, p. 571–572.

[25] D. Spadini, M. Aniche, M. Bruntink, A. Bacchelli, Mock objects for testing java systems, Empirical Software Engineering 24 (3) (2018) 1461–1498. `doi:10.1007/s10664-018-9663-0`.

[26] A. B. Johnston, D. C. Burnett, WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web, Digital Codex LLC, 2012.

[27] M. R. Hoffmann, B. Janiczak, E. Mandrikov, M. Friedenhagen, Jacoco code coverage library (Apr. 2022).

[28] J. Laski, W. Szermer, Identification of program modifications and its applications in software maintenance, in: Proceedings Conference on Software Maintenance 1992, IEEE Computer Society, 1992, pp. 282–283.

[29] G. Rothermel, M. J. Harrold, A safe, efficient regression test selection technique, ACM Transactions on Software Engineering and Methodology (TOSEM) 6 (2) (1997) 173–210.

[30] R. Gupta, M. J. Harrold, M. L. Soffa, An approach to regression testing using slicing., in: ICSM, Vol. 92, 1992, pp. 299–308.

[31] S. S. Yau, Z. Kishimoto, Method for revalidating modified programs in the maintenance phase., in: Proceedings-IEEE Computer Society's International Computer Software & Applications Conference, IEEE, 1987, pp. 272–277.

[32] F. I. Vokolos, P. G. Frankl, Pythia: A regression test selection tool based on textual differencing, in: Reliability, quality and safety of software-intensive systems, Springer, 1997, pp. 3–21.

[33] M. Waseem, P. Liang, G. Márquez, A. D. Salle, Testing microservices architecture-based applications: A systematic mapping study, in: Proc. of APSEC, IEEE, 2020, pp. 119–128.

[34] J. P. Sotomayor, S. C. Allala, D. Santiago, T. M. King, P. J. Clarke, Comparison of open-source runtime testing tools for microservices, Software Quality Journal (May 2022). `doi:10.1007/s11219-022-09583-4`.
URL `https://doi.org/10.1007/s11219-022-09583-4`

[35] K. Meinke, P. Nycander, Learning-based testing of distributed microservice architectures: Correctness and fault injection, in: Proc. of SEFM 2015 Collocated Workshops, Vol. 9509 of LNCS, Springer, 2015, pp. 3–10.

[36] J. G. Quenum, S. Aknine, Towards executable specifications for microservices, in: Proc. of SCC, IEEE, 2018, pp. 41–48.

[37] L. M. Hillah, A.-P. Maesano, F. De Rosa, F. Kordon, P.-H. Wuillemin, R. Fontanelli, S. D. Bona, D. Guerri, L. Maesano, Automation and intelligent scheduling of distributed system functional testing, International Journal on Software Tools for Technology Transfer 19 (3) (2017) 281–308. `doi:10.1007/s10009-016-0440-3`.
URL `https://doi.org/10.1007/s10009-016-0440-3`

[38] M. Camilli, C. Bellettini, L. Capra, M. Monga, A formal framework for specifying and verifying microservices based process flows, in: A. Cerone, M. Roveri (Eds.), Software Engineering and Formal Methods, Springer International Publishing, Cham, 2018, pp. 187–202.

[39] M. Rahman, Z. Chen, J. Gao, A service framework for parallel test execution on a developer's local development workstation, in: 2015 IEEE Symposium on Service-Oriented System Engineering, 2015, pp. 153–160. `doi:10.1109/SOSE.2015.45`.

[40] S.-P. Ma, C.-Y. Fan, Y. Chuang, I.-H. Liu, C.-W. Lan, Graph-based and scenario-driven microservice analysis, retrieval, and testing, Future Generation Computer Systems 100 (2019) 724–735. `doi:https://doi.org/10.1016/j.future.2019.05.048`.
URL `https://www.sciencedirect.com/science/article/pii/S0167739X19302614`

[41] H. Schulz, T. Angerstein, D. Okanović, A. van Hoorn, Microservice-tailored generation of session-based workload models for representative load testing, in: 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2019, pp. 323–335. `doi:10.1109/MASCOTS.2019.00043`.

[42] A. Walenstein, M. El-Ramly, J. R. Cordy, W. S. Evans, K. Mahdavi, M. Pizka, G. Ramalingam, J. W. von Gudenberg, Similarity in programs, in: R. Koschke, E. Merlo, A. Walenstein (Eds.), Duplication, Redundancy, and Similarity in Software, Vol. 6301 of Dagstuhl Seminar Proceedings, Schloss Dagstuhl, Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2007, pp. 1–8. `doi:10.4230/DagSemProc.06301.11`.

[43] A. Sheneamer, J. Kalita, A Survey of Software Clone Detection Techniques, International Journal of Computer Applications 137 (10) (2016) 1–21. `doi:10.5120/ijca2016908896`.
URL `http://www.ijcaonline.org/research/volume137/`

number10/sheneamer-2016-ijca-908896.pdf

[44] T. Lancaster, F. Culwin, A Comparison of Source Code Plagiarism Detection Engines, Computer Science Education 14 (2) (2004) 101–112. doi:10.1080/08993400412331363843.
URL http://www.tandfonline.com/doi/abs/10.1080/08993400412331363843

[45] W. S. Evans, C. W. Fraser, Grammar-based compression of interpreted code, Communications of the ACM 46 (8) (2003) 61–66. doi:10.1145/859670.859699.
URL https://dl.acm.org/doi/10.1145/859670.859699

[46] T. B. Noor, H. Hemmati, A similarity-based approach for test case prioritization using historical failure data, in: Proceedings of the 26th International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2015, pp. 58–68. doi:10.1109/ISSRE.2015.7381799.
URL http://ieeexplore.ieee.org/document/7381799/

[47] R. Wang, S. Jiang, D. Chen, Similarity-based regression test case prioritization, in: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, KSI Research Inc., 2015, p. 6. doi:10.18293/SEKE2015-115.

[48] D. Hao, L. Zhang, Y. Pan, H. Mei, J. Sun, On similarity-awareness in testing-based fault localization, Automated Software Engineering 15 (2) (2008) 207–249. doi:10.1007/s10515-008-0025-9.
URL http://link.springer.com/10.1007/s10515-008-0025-9

[49] Y. Ledru, A. Petrenko, S. Boroday, N. Mandran, Prioritizing test cases with string distances, Autom. Softw. Eng. 19 (1) (2012) 65–95. doi:10.1007/s10515-011-0093-0.

[50] B. Miranda, E. Cruciani, R. Verdecchia, A. Bertolino, FAST approaches to scalable similarity-based test case prioritization, in: M. Chaudron, I. Crnkovic, M. Chechik, M. Harman (Eds.), Proceedings of the 40th International Conference on Software Engineering, ACM, 2018, pp. 222–232. doi:10.1145/3180155.3180210.

[51] D. Raman, B. Bezawada, T. V. Rajinikanth, S. Sathyanarayan, Static Program Behavior Tracing for Program Similarity Quantification, in: S. C. Satapathy, V. K. Prasad, B. P. Rani, S. K. Udgata, K. S. Raju (Eds.), Proceedings of the First International Conference on Computational Intelligence and Informatics, Vol. 507 of Advances in Intelligent Systems and Computing (AISC), Springer, Singapore, 2017, pp. 321–330. doi:10.1007/978-981-10-2471-9{_}31.
URL https://link.springer.com/chapter/10.1007/978-981-10-2471-9_31

[52] S. Wang, D. Wu, In-memory fuzzing for binary code similarity analysis, in: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, 2017, pp. 319–330. doi:10.1109/ASE.2017.8115645.
URL http://ieeexplore.ieee.org/document/8115645/

[53] C. Meudec, ATGen: automatic test data generation using constraint logic programming and symbolic execution†, Software Testing, Verification and Reliability 11 (2) (2001) 81–96.

[54] W. Visser, C. S. Pasareanu, S. Khurshid, Test input generation with java pathfinder, in: Proc. of ISSTA, ACM, 2004, pp. 97–107.

[55] C. Cadar, D. R. Engler, Execution generated test cases: How to make systems code crash itself, in: Proc. of Int. Workshop SPIN, Vol. 3639 of LNCS, Springer, 2005, pp. 2–23.

[56] P. Godefroid, N. Klarlund, K. Sen, DART: Directed Automated Random Testing, in: Proc. of PLDI, ACM, 2005, pp. 213–223. doi:10.1145/1065010.1065036.

[57] P. Braione, G. Denaro, A. Mattavelli, M. Pezzè, SUSHI: a test generator for programs with complex structured inputs, in: Proc. of ICSE, ACM, 2018, pp. 21–24.

[58] K. Nivethithaa, V. Krishnapriya, A Brief Survey on Symbolic Execution Test-Selection Techniques, International Journal of Computer Sciences and Engineering 06 (08) (2018) 81–85. doi:10.26438/ijcse/v6si8.8185.
URL http://www.ijcseonline.org/full_spl_paper_view.php?paper_id=480

[59] A. Armando, J. Mantovani, L. Platania, Bounded model checking of software using SMT solvers instead of SAT solvers, Int. J. Softw. Tools Technol. Transf. 11 (1) (2009) 69–83.

[60] N. Williams, B. Marre, P. Mouy, M. Roger, PathCrawler: Automatic generation of path tests by combining static and dynamic analysis, in: Proc. of EDCC, Vol. 3463 of LNCS, Springer, 2005, pp. 281–292.

[61] D. Felsing, S. Grebing, V. Klebanov, P. Rümmer, M. Ulbrich, Automating regression verification, in: Proc. of ASE, 2014, pp. 349–360.

[62] E. De Angelis, F. Fioravanti, A. Pettorossi, M. Proietti, Relational verification through horn clause transformation, in: Proc. of SAS, Vol. 9837 of LNCS, Springer, 2016, pp. 147–169.

[63] P. Collingbourne, C. Cadar, P. H. J. Kelly, Symbolic crosschecking of data-parallel floating-point code, IEEE Trans. Software Eng. 40 (7) (2014) 710–737.

Appendix A. Implementation Details and Further Results

Appendix A.1. Test Program Enumeration

As already mentioned, our analysis technique assumes that test programs are clearly identifiable from the rest of the source code—we explicitly consider JUnit 4/5 annotations.

Our test program enumeration mechanism is based on a JSON configuration file, the structure of which is reported in Listing 10. For test program discovery, we rely on the `testProgram` list, which allows the user to specify the paths to the folders where the compiled test classes can be found on disk. We scan these paths and recursively enumerate all classes. In each class found, all methods annotated as `@Test` (but without the `@Ignore` annotation) are located and stored in an in-memory dictionary.

The possibility of relying on a list of paths is significant in the case of microservices applications. For example, each microservice could have its own unit or integration tests, while contract/end-to-end tests may also be located in different repositories. Allowing the automatic detection of test programs in any project structure is essential in such possibly-complex application organizations.

If for any reason, the QA Engineering team wants to exclude some test programs, this can be done by relying on the `excludeTest` list in the JSON configuration file. Conversely, if only a certain number of test programs should be analyzed, the `includeTest` list in the JSON file can be configured accordingly.

```json
{
  "sut": [
    "path to classes 1",
    "path to classes 2"
  ],
  "testPrograms": [
    "path to test classes 1",
    "path to test classes 1"
  ],
  "includeTest": [
    "list", "of", "@Test",
    "methods", "to", "analyze"
  ],
  "excludeTest": [
    "list", "of", "@Test",
    "methods", "to", "skip"
  ],
  "additionalClasspath": [
    "path", "to", "any",
    "other", "needed", "dependency"
  ],
  "excludeTracedPackages": [
    "java/",
    "sun/"
  ]
}
```

Listing 10: Symbolic Execution JSON Configuration File.

Appendix A.2. Information Extraction via Concrete Execution

To collect behavioural information in a concrete execution, we rely on the *Java Agent* technology.

To collect behavioural information in a concrete execution, we rely on a *java agent*. The java agent exploits a `premain` method registered in the `MANIFEST.MF` file, and takes control upon application startup. In the agent, we register a custom `ClassFileTransformer` by relying on the standard JAVA `java.lang.instrument.Instrumentation` interface.

The agent is directly attached to the JUnit run, relying on the Maven Surefire plugin. The `ClassFileTransformer` receives the bytecode of every class loaded while the tests are running. The agent can be configured to trace only classes belonging to a specific set of package prefixes, while other prefixes can be used to exclude other classes. If a class should be instrumented (i.e., if it belongs to the test program or the SUT), we inject probes at particular points of interest, as we shall discuss.

We rely on Javassist [21] to perform bytecode manipulation. The first step to instrument test programs is to transform the bytecode passed to the `ClassFileTransformer` into a representation suitable for manipulation with Javassist. This is done according to the code snippet reported in Listing 11. We explicitly rely on the classloader passed to the transformation method to account for the location of the original class (`loader`, at line 2). We also explicitly import the agent's package where the probes are implemented (line 3). Then, we try to transform the bytecode into a `CtClass` (compile-time class) object, which is a Javassist handle for dealing with a class file (line 7).

We then enumerate all methods in the class. For all methods that are not native and are not abstract, we apply the instrumentation scheme depicted in Listing 12. First, we determine whether the method bears the `@Test` annotation for both JUnit 4 and 5 (lines

```
1  ClassPool pool = new ClassPool();
2  pool.appendClassPath(new LoaderClassPath(loader));
3  pool.importPackage("it.cnr.iasi.saks.inspection");
4
5  CtClass ctClass;
6  try {
7      ctClass = pool.makeClass(new ByteArrayInputStream(classfileBuffer));
8  } catch (IOException e) {
9      log.error("Unable to build CtClass for class: {}", clazzName);
10     e.printStackTrace();
11     return null;
12 }
13 assert ctClass != null;
```

Listing 11: Build a dedicated class pool for this instrumentation and import required packages.

```
1  Object annotation4 = null;
2  Object annotation5 = null;
3  try {
4      annotation4 = method.getAnnotation(org.junit.Test.class);
5      annotation5 = method.getAnnotation(org.junit.jupiter.api.Test.class);
6  } catch (ClassNotFoundException ignored) {}
7
8  // Insert the instrumentation code at the start/end of the method.
9  try {
10     if(annotation4 != null || annotation5 != null) {
11         method.insertBefore("MetricsCollector.instance().enterTest(\"" + clazzName + "\",\"" + method.getName() + "\");");
12         method.insertAfter("MetricsCollector.instance().leaveTest(\"" + clazzName + "\",\"" + method.getName() + "\");");
13     } else {
14         method.insertBefore("MetricsCollector.instance().enterMethod(\"" + clazzName + "\",\"" + method.getName() + "\",\"" +
               method.getMethodInfo().getDescriptor() + "\", $args);");
15         method.insertAfter("MetricsCollector.instance().leaveMethod(\"" + clazzName + "\",\"" + method.getName() + "\",\"" + method
               .getMethodInfo().getDescriptor() + "\");");
16     }
17 } catch (CannotCompileException e) {
18     log.error("Unable to instrument class {}. The class has not been instrumented.", clazzName);
19     e.printStackTrace();
20     return null;
21 }
```

Listing 12: Method Instrumentation.

1–6). In the positive case, the method is the entry point of a test program. We thus inject a method call to our custom methods `enterTest()` and `leaveTest()` at the beginning and the end of the method, respectively. In the negative case, the method is any other method in the test program (or in the SUT). We thus add at the beginning and end of the method calls to `enterMethod()`/ `leaveMethod()` respectively.

These methods are located in the agent's package `it.cnr.iasi.saks.inspection` that was imported before (Listing 11, line 3). Calls to `enterMethod()` and `leaveMethod()` allow us to build an in-memory representation of the Prolog facts described above, while calls to `enterTest()` and `leaveTest()` allow us to associate these methods with specific test programs. When a test program ends, the in-memory representation is dumped to a disk file, respecting the `invokes` format described above. All elements related to symbolic execution are set to placeholder values.

Appendix A.3. The `filter` *predicate*

This section presents the implementation details of the `filter` predicate and how to use it to process `invokes` facts.

The predicate `filter(Xs,XSchema,XSelFun,XExtFun,YName,Ys)`, shown in Listing 13, takes as input any given list of Prolog facts and generates a list of terms `Ys` satisfying the following conditions. Any element `Y` of `Ys`: (i) is a term with functor `YName`, (ii) is obtained from a term `X` in `Xs` for which all predicates in the list `XSelFun` hold (that is, `XSelFun` is a list of functions for selecting elements from `Xs`), and (iii) the *i*-th argument of `Y` is obtained by applying to `X` the *i*-th predicate in the list `XExtFun` (that is, `XExtFun` is a list of functions for generating the arguments of `Y` using the information extracted from the selected `X`). The second argument `XSchema` of `filter` is a ground term that defines the structure of any term in `Xs` by assigning labels to its arguments. The mapping between the arguments of `XSchema` and the arguments of terms in `Xs` provides an easy way to get any argument of a term in `Xs` by using the corresponding label in `XSchema`.

In particular, the recursive rules of `filter`, that is, the rules starting at lines 2 and 7 in Listing 13, take the head `X` of the input list and perform the following operations. The predicate `eval_sel_fun` in the second rule of `filter` checks if the predicates in `XSelFun` hold for `X`. If so, the predicate `eval_ext_fun` generates the list `YArgs` from `X` and makes use of the Prolog "univ" operator "`=..`" (line 5) to construct a new term `Y`, named `YName`, whose arguments are the terms in `YArgs`; otherwise the third rule

```
1  filter([],XSchema,XSelFun,XExtFun,YName,[]).
2  filter([X|Xs],XSchema,XSelFun,XExtFun,YName,[Y|Ys]) :-
3      eval_sel_fun(X,XSchema,XSelFun), !,
4      eval_ext_fun(X,XSchema,XExtFun,YArgs),
5      Y =.. [YName|YArgs],
6      filter(Xs,XSchema,XSelFun,XExtFun,YName,Ys).
7  filter([X|Xs],XSchema,XSelFun,XExtFun,YName,Ys) :-
8      filter(Xs,XSchema,XSelFun,XExtFun,YName,Ys).
```

Listing 13: Prolog rule that defines `filter(Xs,XSchema,XSelFun,XExtFun,YName,Ys)`.

of `filter` applies and `X` is ignored. The Prolog "cut" predicate "`!`" (line 3) prevents the application of the third rule whenever the second rule applies.

The code snippet in Listing 14 shows how to select the `invokes` facts whose caller and callee components do not belong to some given lists. The first parameter `InvokesLst` (line 2) is the list of `invokes` facts, whose structure is specified as the second parameter (line 3). The third parameter makes use of the utility predicate `notIn(Element,List)` (line 4) to select all `invokes` such that: (i) `notIn(caller,CallerBLst)` holds, that is, the caller does not belong to the given list of callers `CallerBLst`, and (ii) `notIn(callee,CalleeBLst)` holds, that is, the callee does not belong to the given list of callees `CalleeBLst`. Using as fourth parameter (line 5) exactly the list of all arguments of the schema specified as the second argument, and as fifth argument the functor `invokes` (line 6), we have that the filter yields exactly the selected `invokes` and adds it to the output list `SelectedLst` (line 7).

```
1  filter(
2    InvokesLst,
3    invokes(testProgram,branchingPointList,seqNum,caller,programPoint,frameEpoch,pathCondition,callee,parameters),
4    [ notIn(caller,CallerBLst), notIn(callee,CalleeBLst) ],
5    [ testProgram, branchingPointList, seqNum, caller, programPoint, frameEpoch, pathCondition, callee, parameters ],
6    invokes,
7    SelectedLst
8   )
```

Listing 14: Prolog query to select `invokes` facts.

The `filter` predicate can also be used to generate new facts by reshaping the data extracted from those occurring in the first argument. In particular, the code snippet in Listing 15 shows the query that performs the operations (1)–(3) described in Section 4.4 to generate `endpoint` facts. The first two parameters are the same as Listing 14. The third parameter is the utility predicate `isHttpMethod(callee)` (line 4), which selects all `invokes` facts whose callees make use of an HTTP method to invoke a remote API. The fourth parameter (line 5) is the list of fields to be extracted from the selected `invokes` facts, specifically:

1. the name of the test program `testProgram`,
2. the caller method `method(caller)`,
3. the HTTP method `httpMethod(callee,parameter)` (which occurs either as part of the callee's name or as a callee's parameter), and
4. the first parameters of the HTTP method `head(parameters)`, that is, the URI of the remote API.

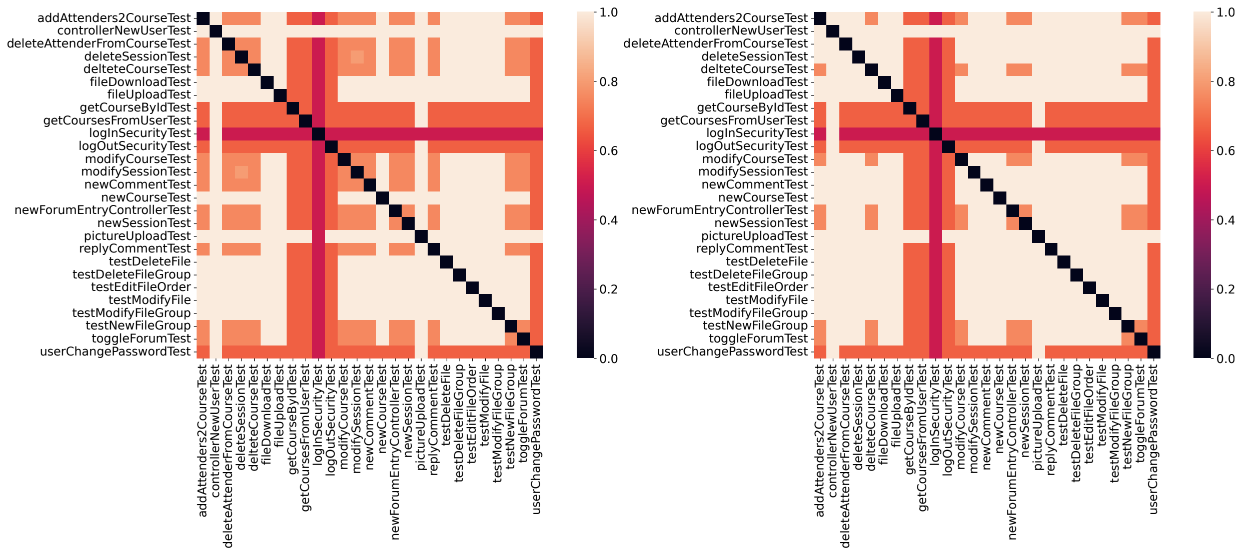
These fields become the arguments of the newly generated fact, called `endpoint` (line 6), which is added to the output list `EndpointLst` (line 7).

```
1  filter(
2    InvokesLst,
3    invokes(testProgram,branchingPointList,seqNum,caller,programPoint,frameEpoch,pathCondition,callee,parameters),
4    [ isHttpMethod(callee) ],
5    [ testProgram, method(caller), httpMethod(callee,parameters), head(parameters) ],
6    endpoint,
7    EndpointLst)
```

Listing 15: Prolog query to generate `endpoint` facts.

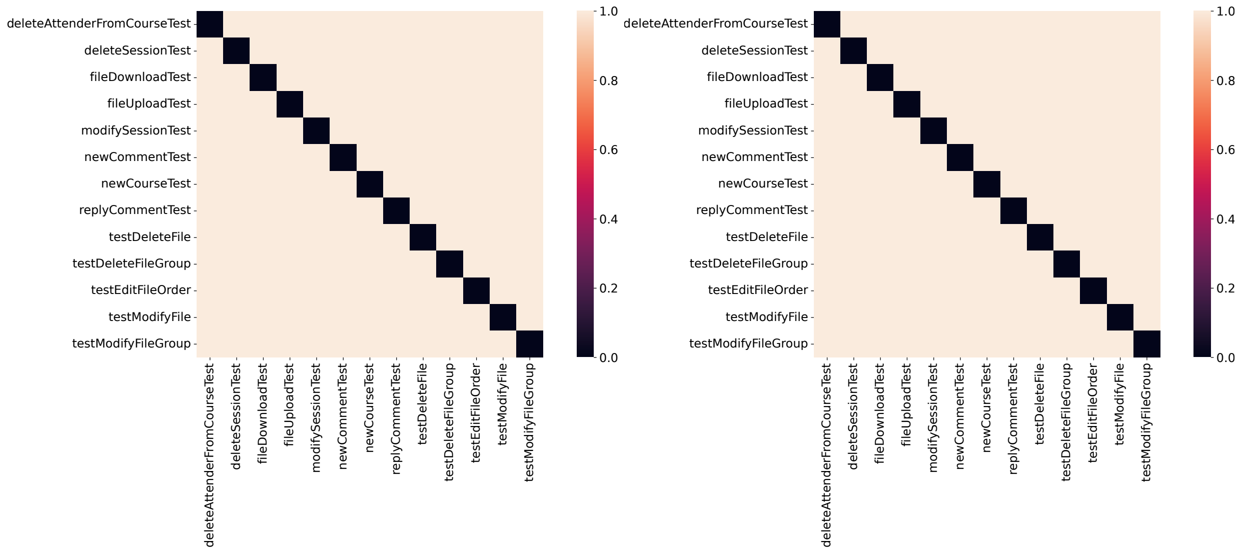
Appendix A.4. Test Program Similarity Heatmaps

Appendix A.4.1. Fullteaching: Symbolic Execution



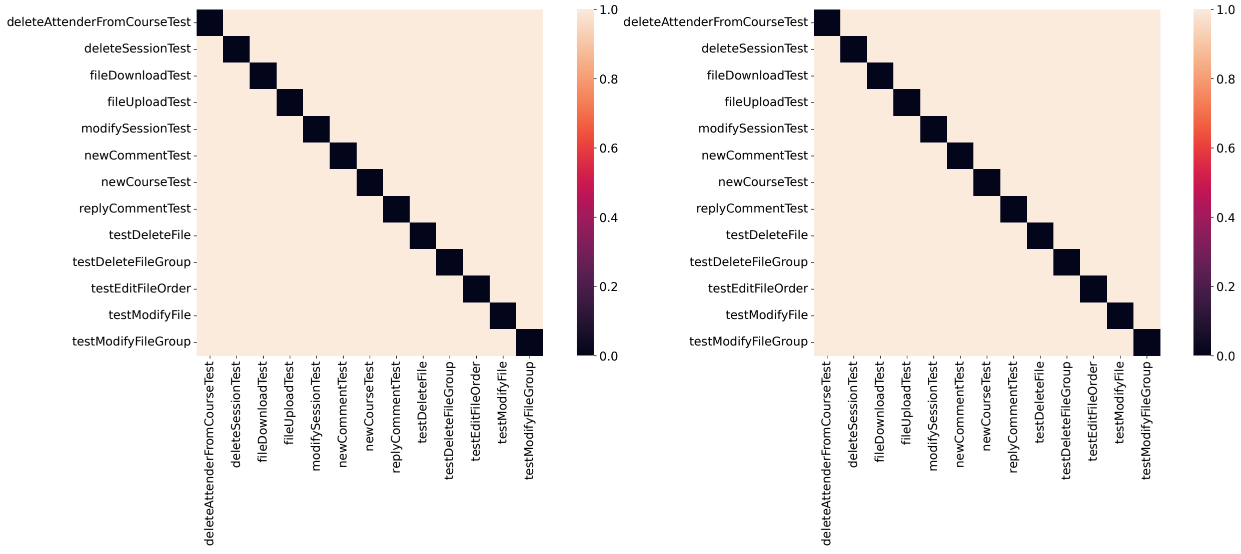
(a) Minimum Similarity Score. (b) Maximum Similarity Score.

Figure A.18: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyCommonSeq`.



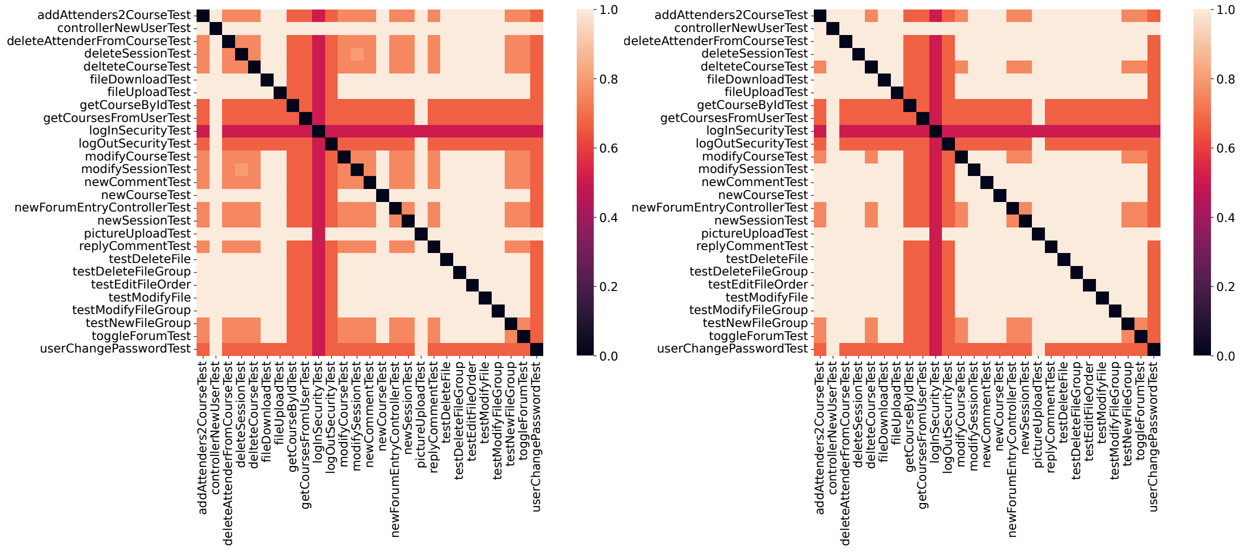
(a) Minimum Similarity Score. (b) Maximum Similarity Score.

Figure A.19: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyEqSeq`.



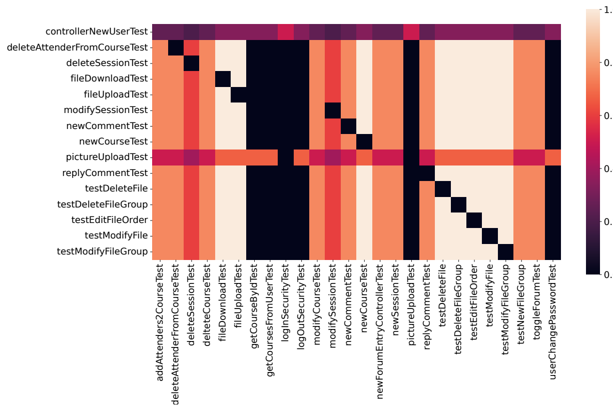
(a) Minimum Similarity Score. (b) Maximum Similarity Score.

Figure A.20: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyEqSet`.



(a) Minimum Similarity Score. (b) Maximum Similarity Score.

Figure A.21: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyIntersection`.


(a) Minimum Similarity Score.

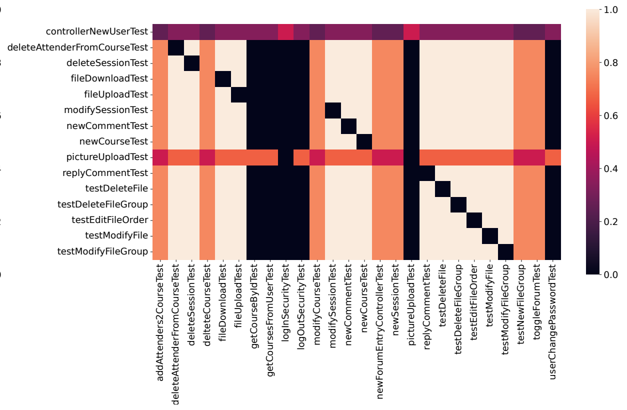
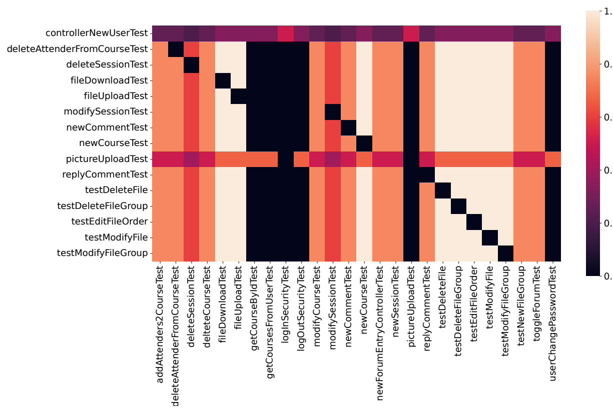

(b) Maximum Similarity Score.

Figure A.22: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptySubSeq`.


(a) Minimum Similarity Score.

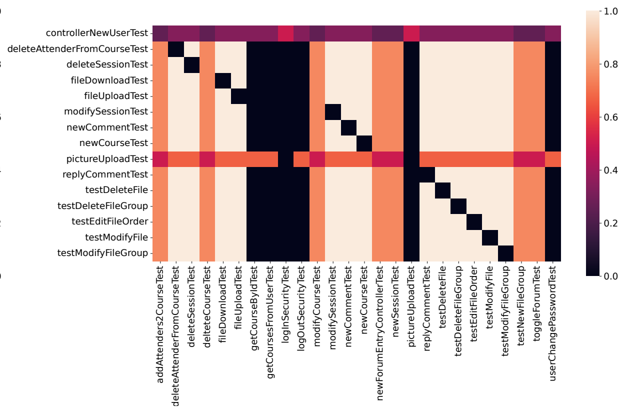
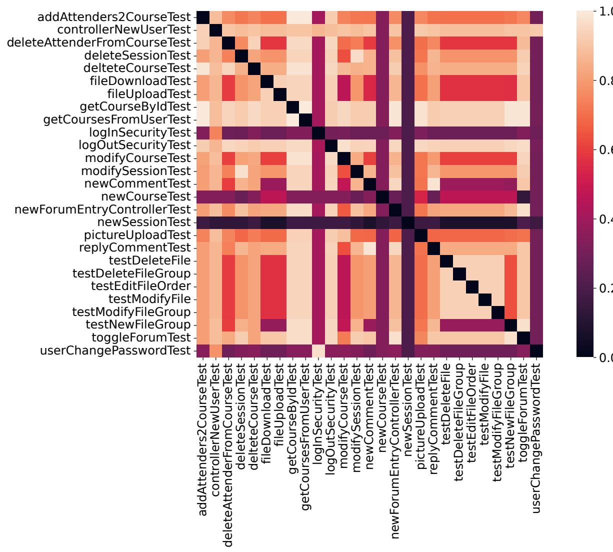

(b) Maximum Similarity Score.

Figure A.23: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptySubSet`.


(a) Minimum Similarity Score.

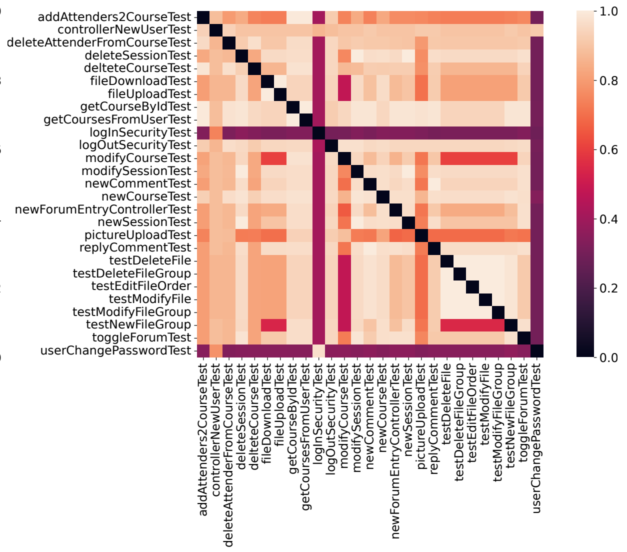
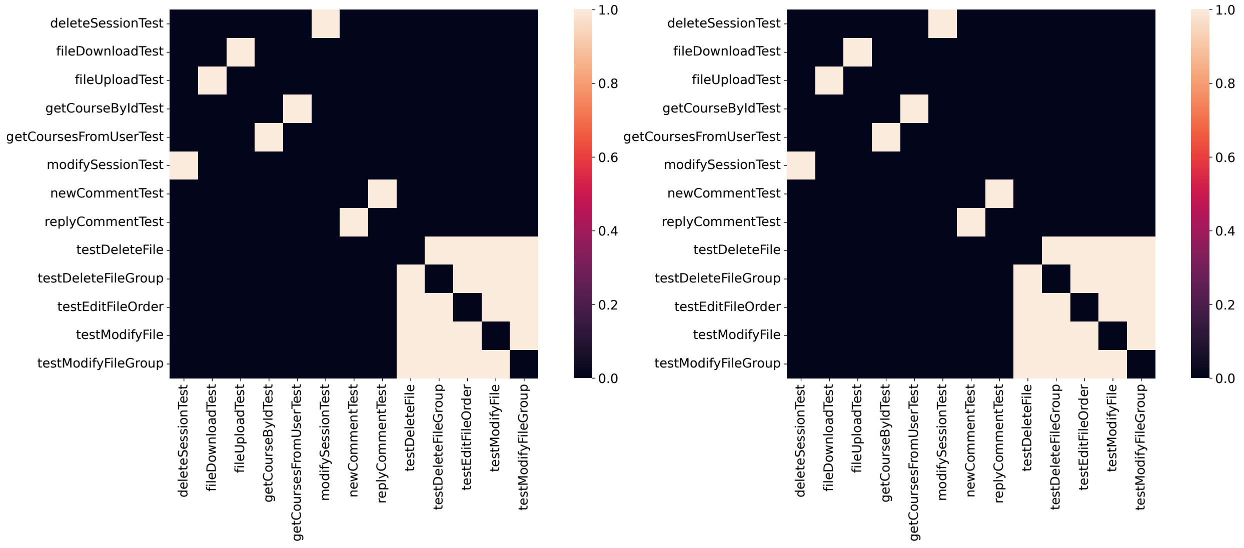
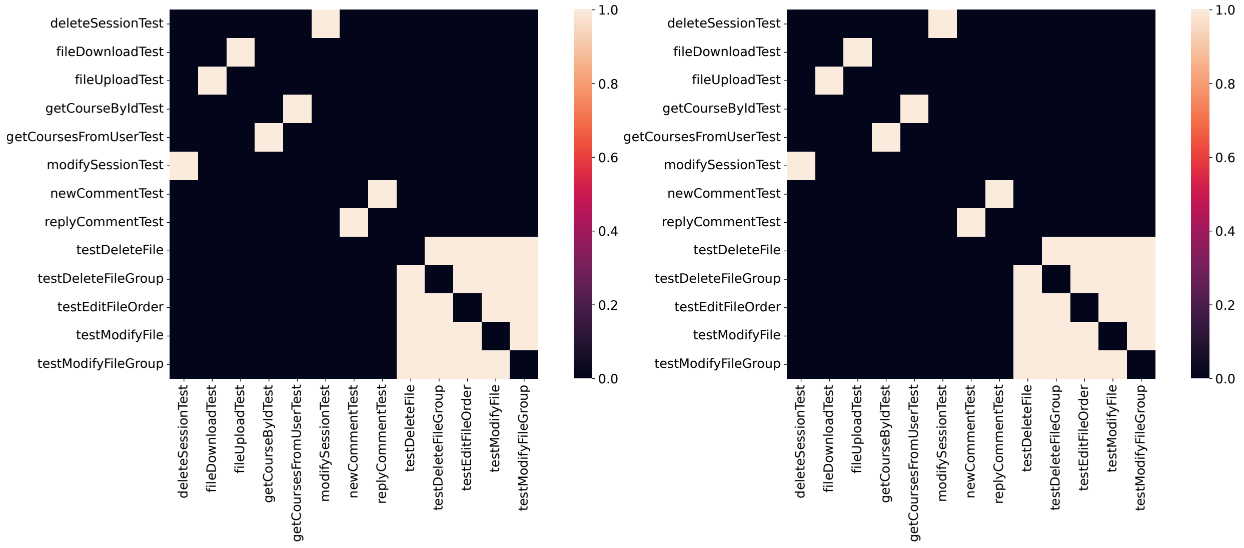

(b) Maximum Similarity Score.

Figure A.24: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptyCommonSeq`.



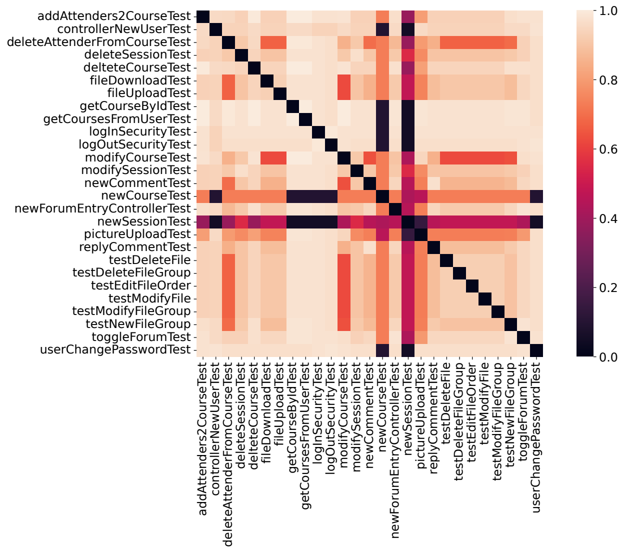
(a) Minimum Similarity Score. (b) Maximum Similarity Score.

Figure A.25: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptyEqSeq`.

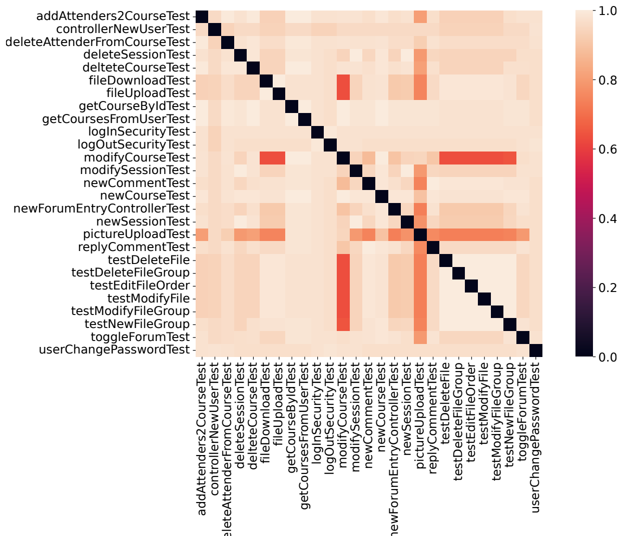


(a) Minimum Similarity Score. (b) Maximum Similarity Score.

Figure A.26: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptyEqSet`.

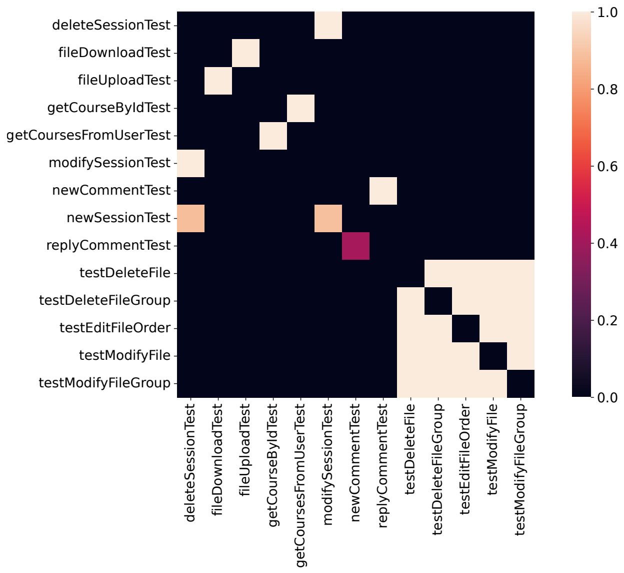


(a) Minimum Similarity Score.

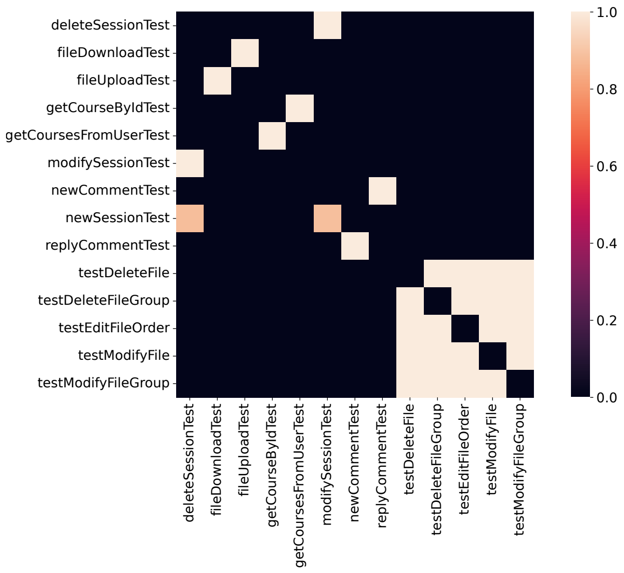


(b) Maximum Similarity Score.

Figure A.27: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptyIntersection`.

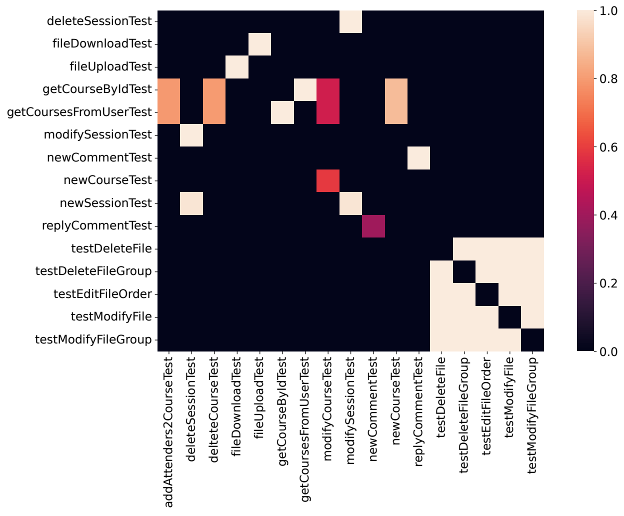


(a) Minimum Similarity Score.

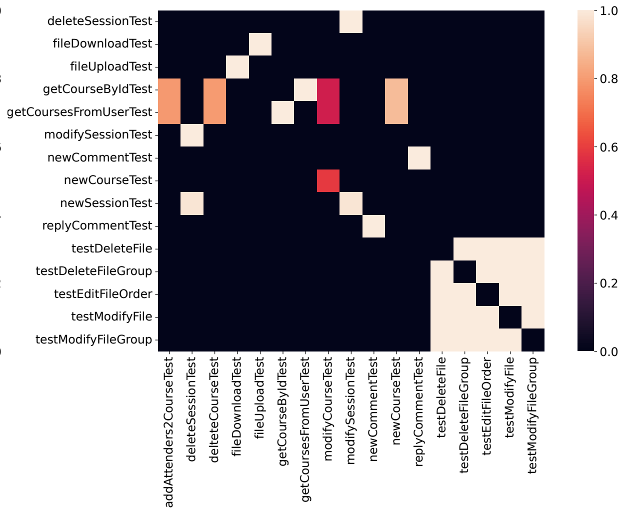


(b) Maximum Similarity Score.

Figure A.28: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptySubSeq`.



(a) Minimum Similarity Score.



(b) Maximum Similarity Score.

Figure A.29: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptySubSet`.

Appendix A.4.2. Fullteaching: Concrete Execution

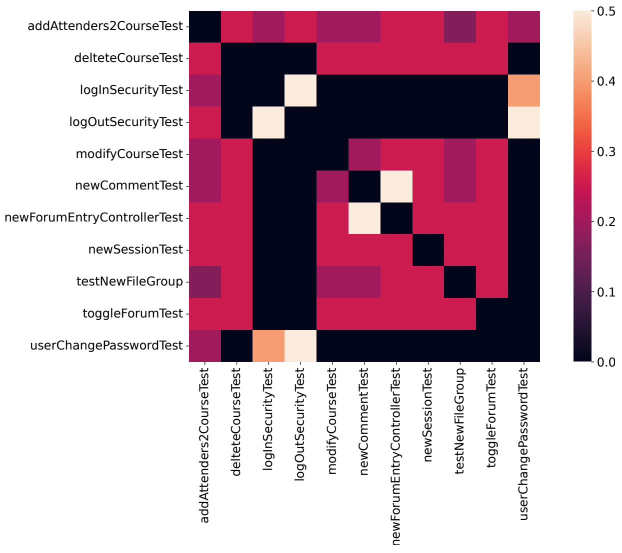


Figure A.30: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyCommonSeq`.

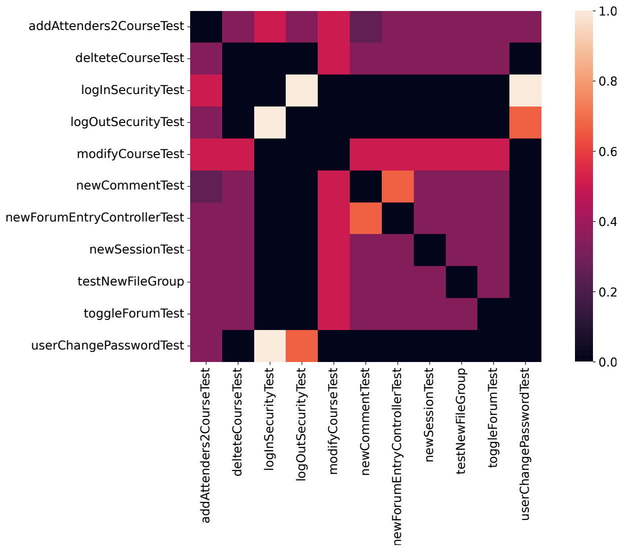


Figure A.31: *Subject*: Fullteaching; *Domain*: `endpoint`; *Criterion*: `nonemptyIntersection`.

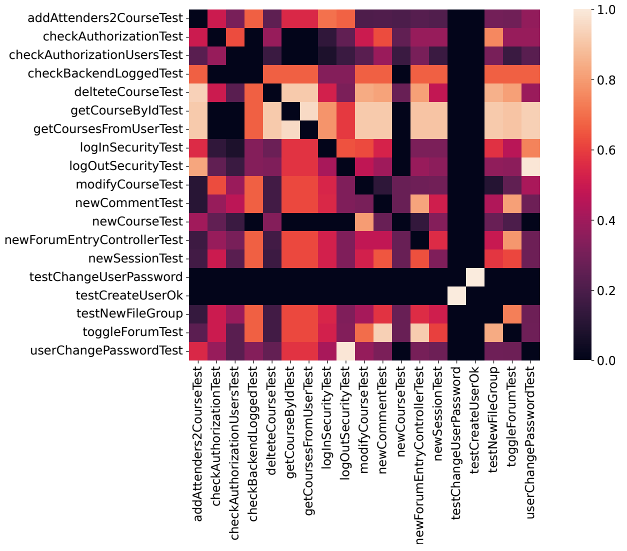


Figure A.32: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptyCommonSeq`.

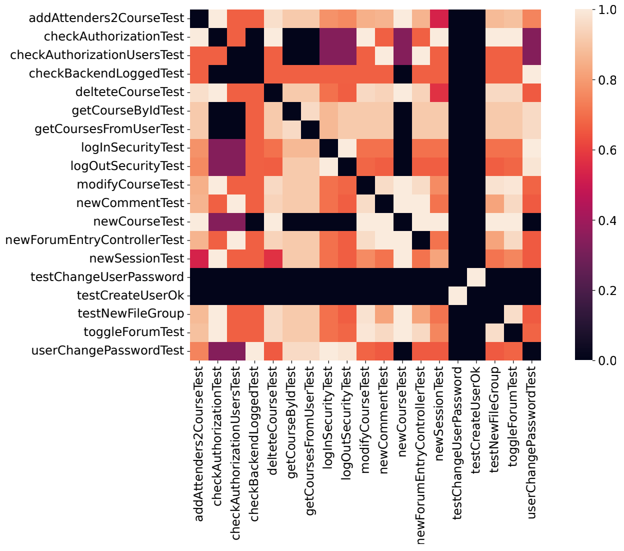


Figure A.33: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptyIntersection`.

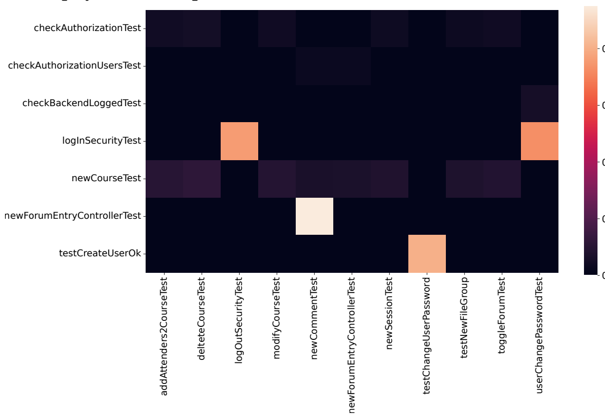
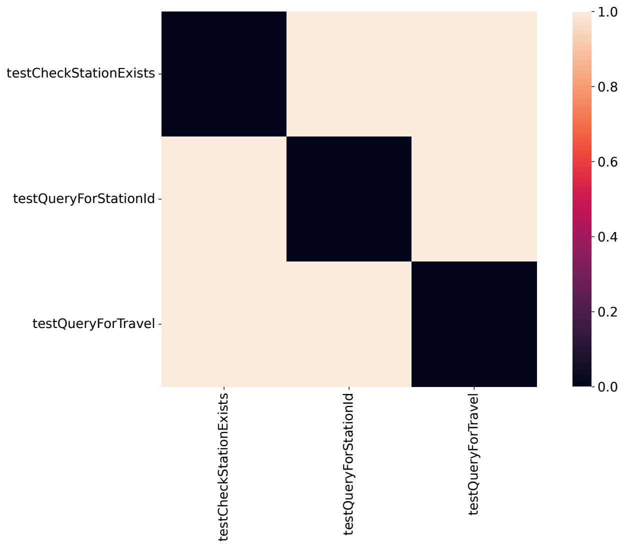
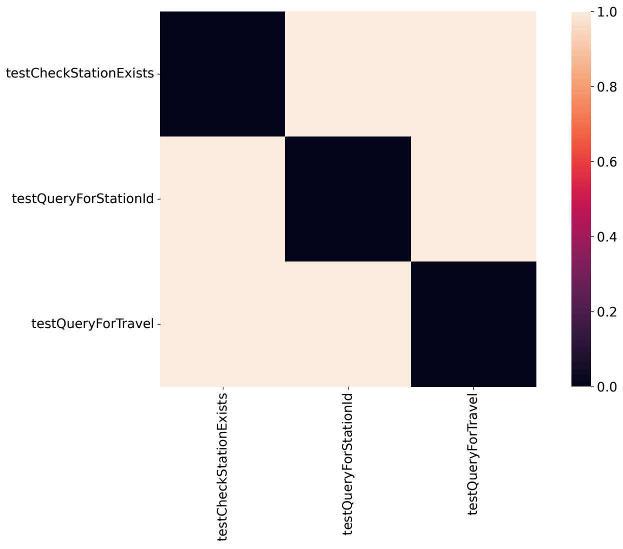


Figure A.34: *Subject*: Fullteaching; *Domain*: `invokes`; *Criterion*: `nonemptySubSet`.

Appendix A.4.3. TrainTicket: Symbolic Execution

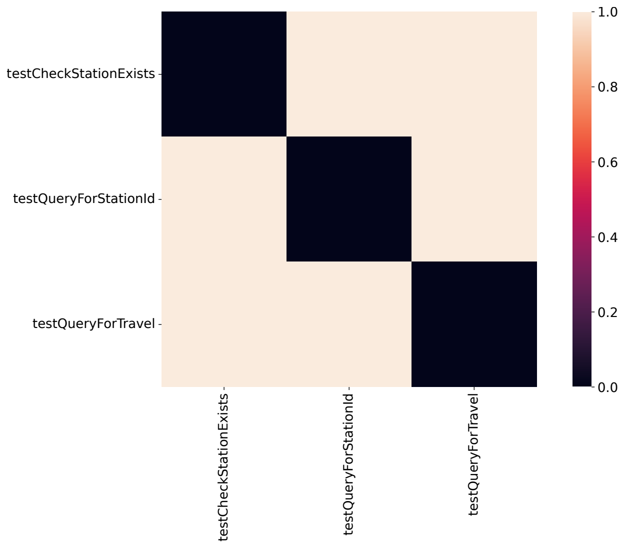


(a) Minimum Similarity Score.

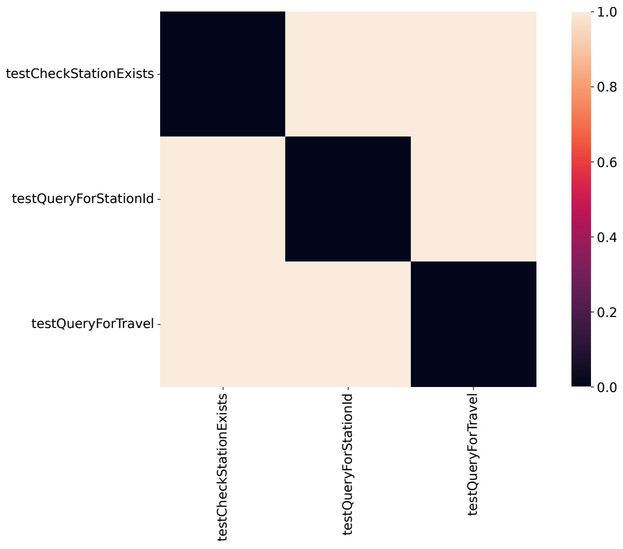


(b) Maximum Similarity Score.

Figure A.35: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyCommonSeq`.

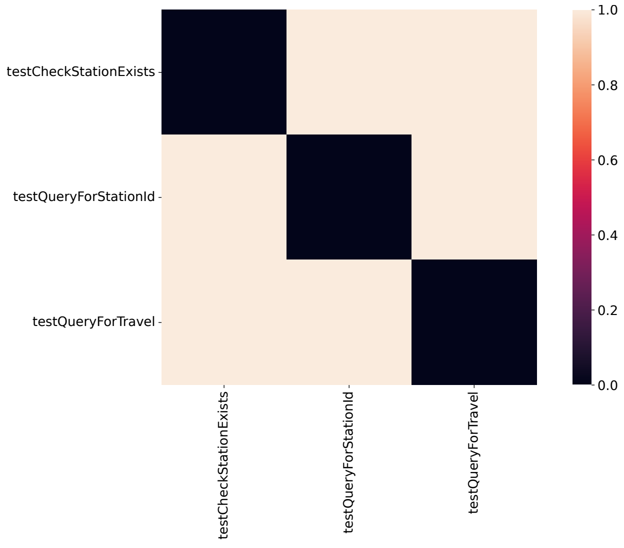


(a) Minimum Similarity Score.

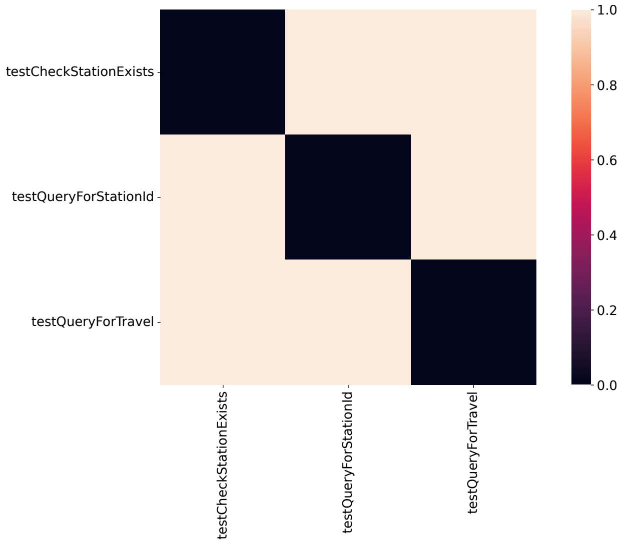


(b) Maximum Similarity Score.

Figure A.36: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyEqSeq`.

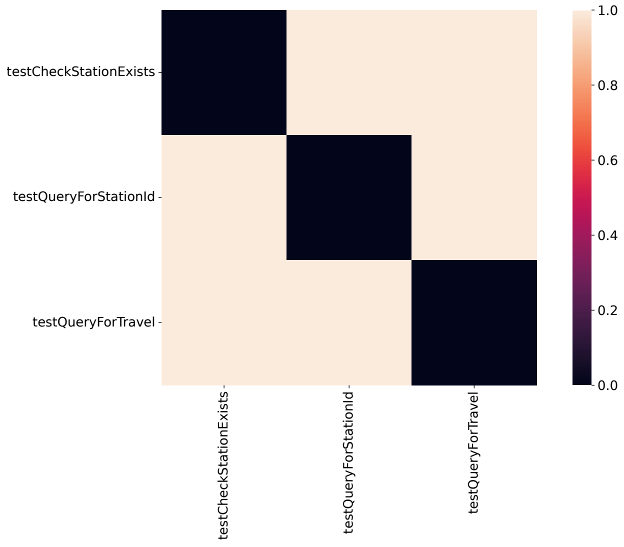


(a) Minimum Similarity Score.

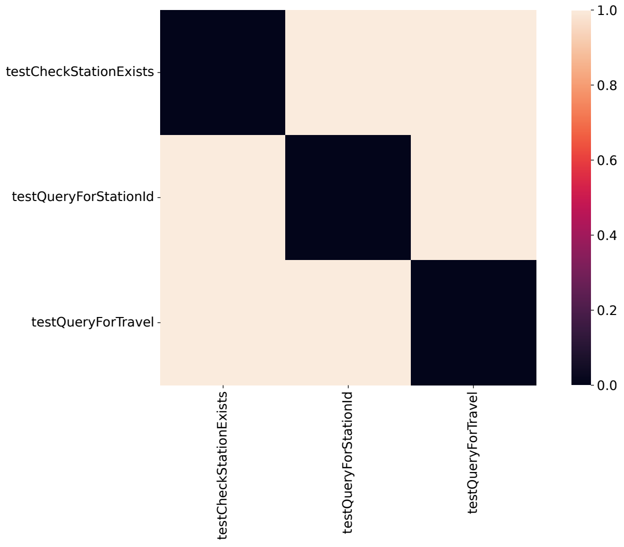


(b) Maximum Similarity Score.

Figure A.37: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyEqSet`.

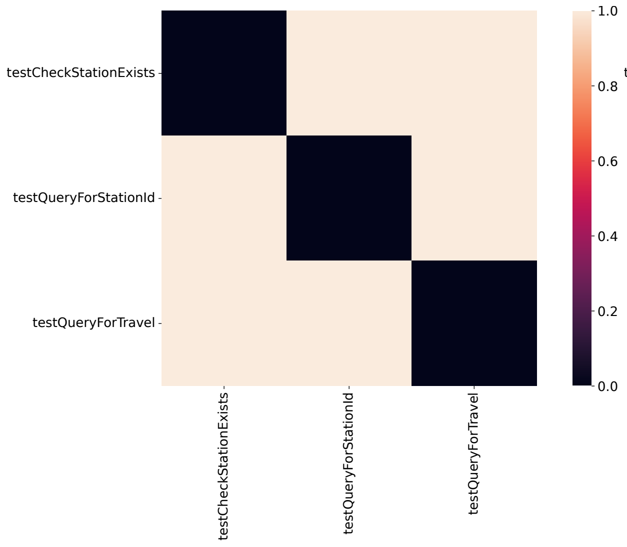


(a) Minimum Similarity Score.

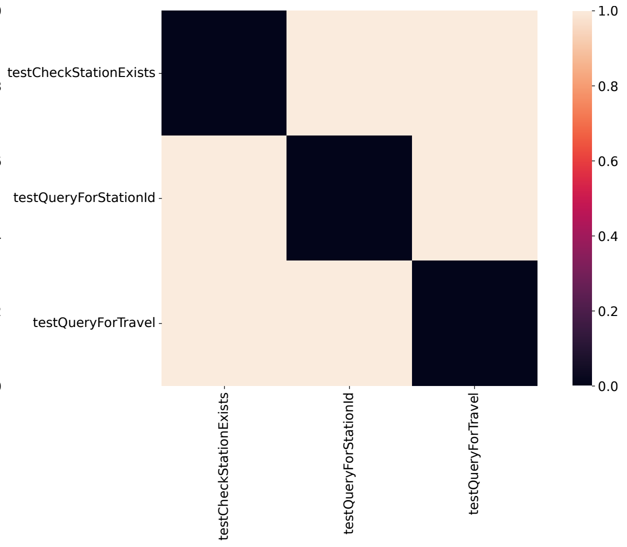


(b) Maximum Similarity Score.

Figure A.38: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyIntersection`.

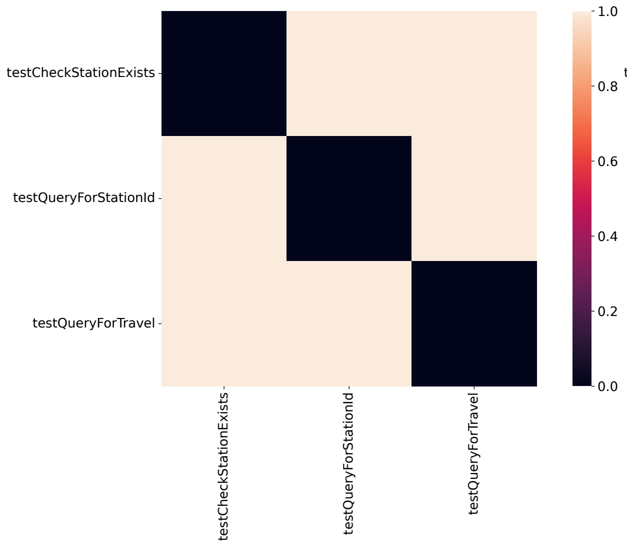


(a) Minimum Similarity Score.

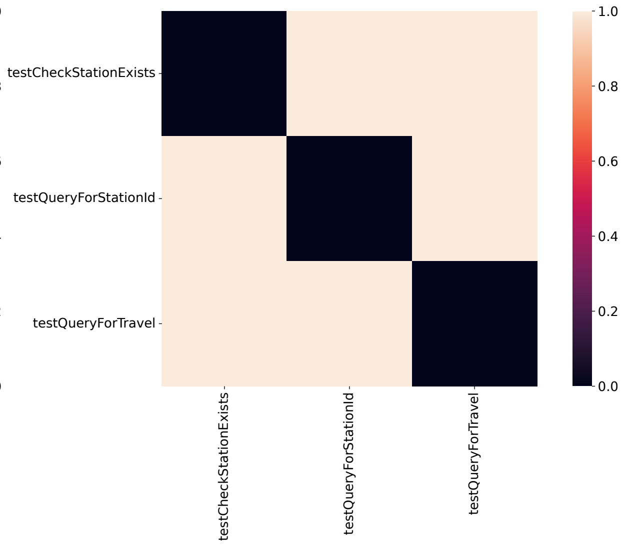


(b) Maximum Similarity Score.

Figure A.39: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptySubSeq`.

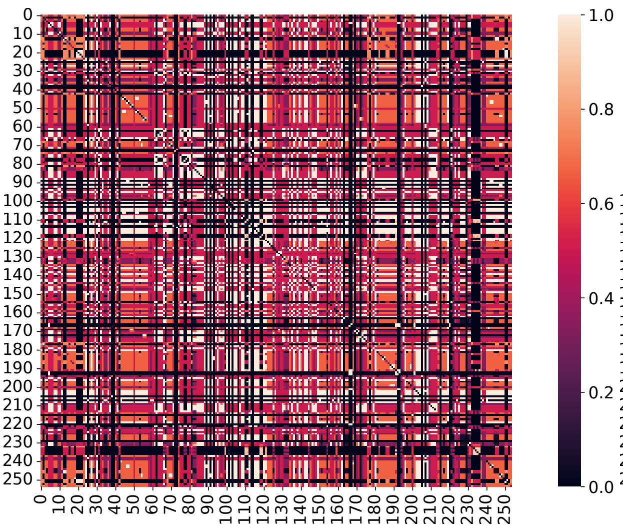


(a) Minimum Similarity Score.



(b) Maximum Similarity Score.

Figure A.40: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptySubSet`.


(a) Minimum Similarity Score.

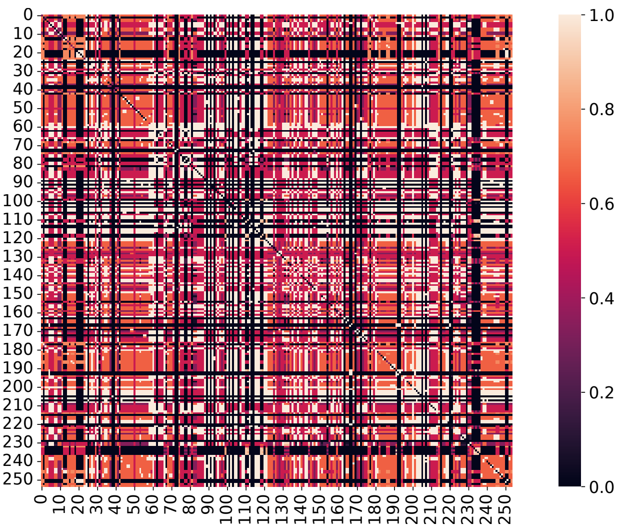
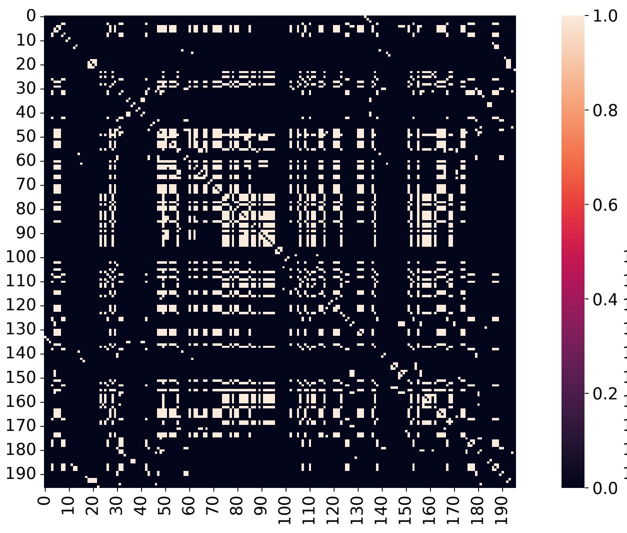

(b) Maximum Similarity Score.

Figure A.41: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyCommonSeq`.


(a) Minimum Similarity Score.

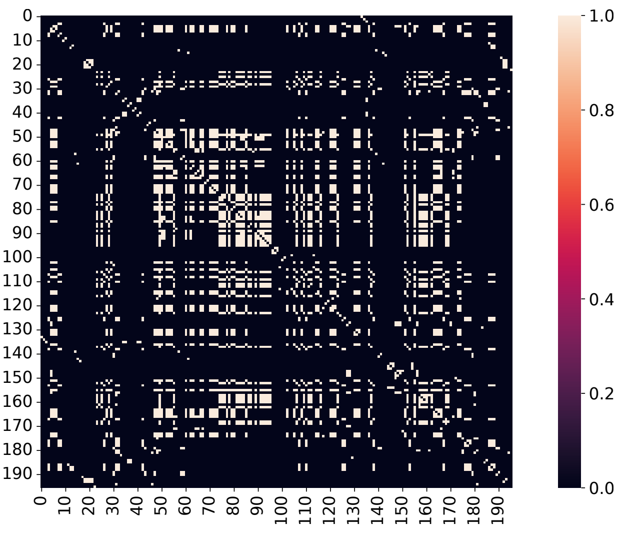
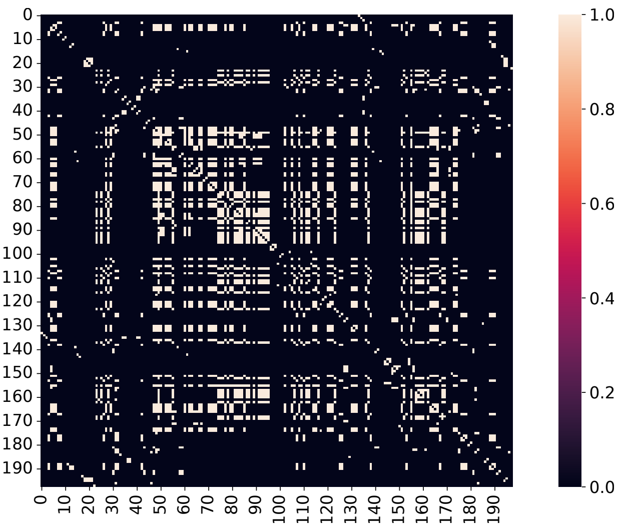

(b) Maximum Similarity Score.

Figure A.42: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyEqSeq`.


(a) Minimum Similarity Score.

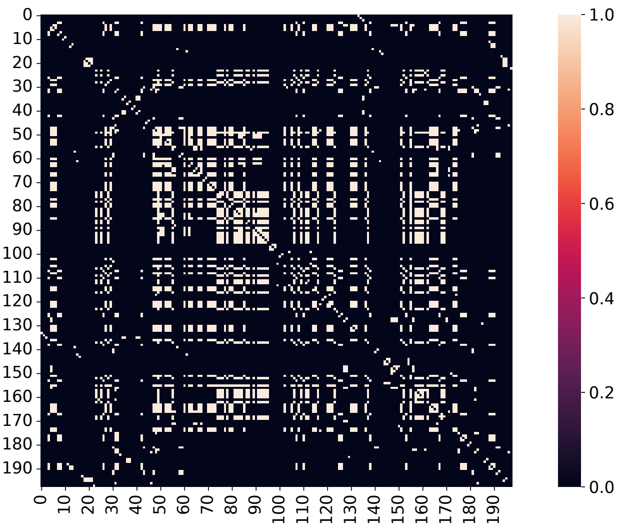
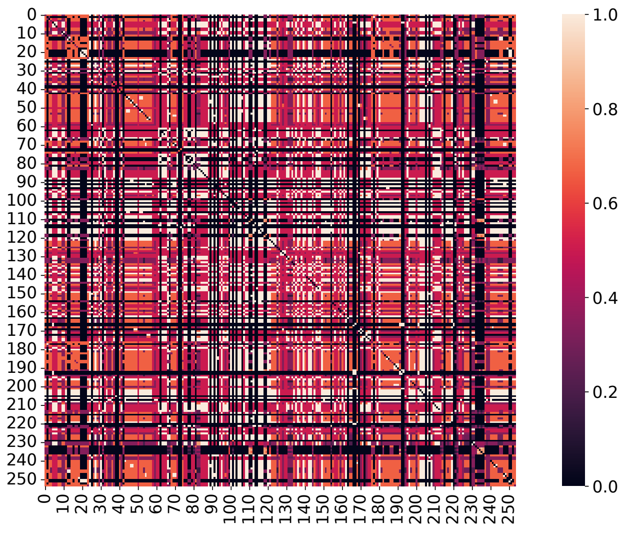

(b) Maximum Similarity Score.

Figure A.43: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyEqSet`.


(a) Minimum Similarity Score.

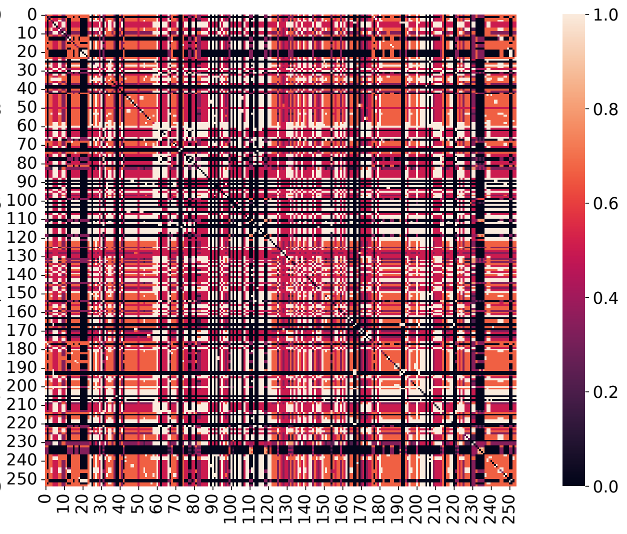
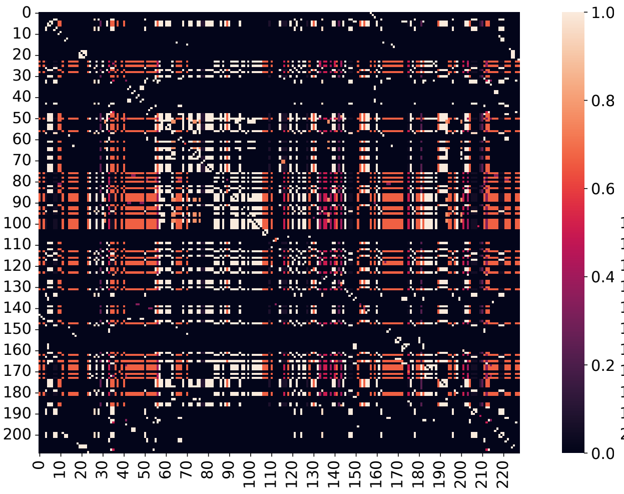
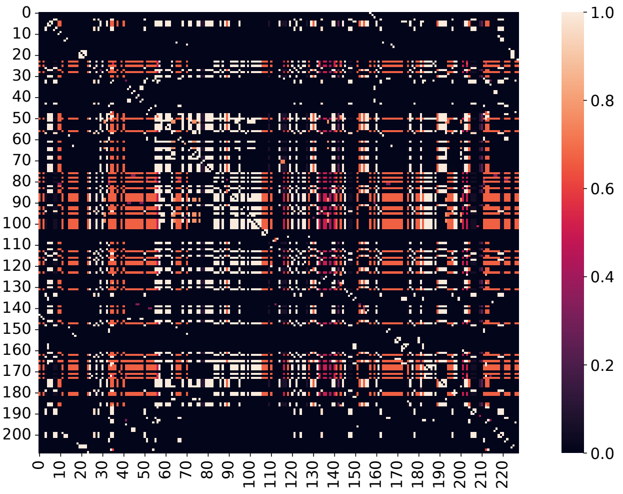

(b) Maximum Similarity Score.

Figure A.44: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyIntersection`.

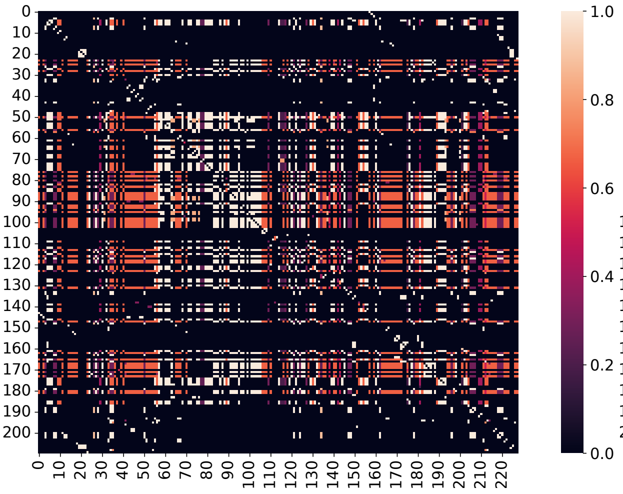


(a) Minimum Similarity Score.

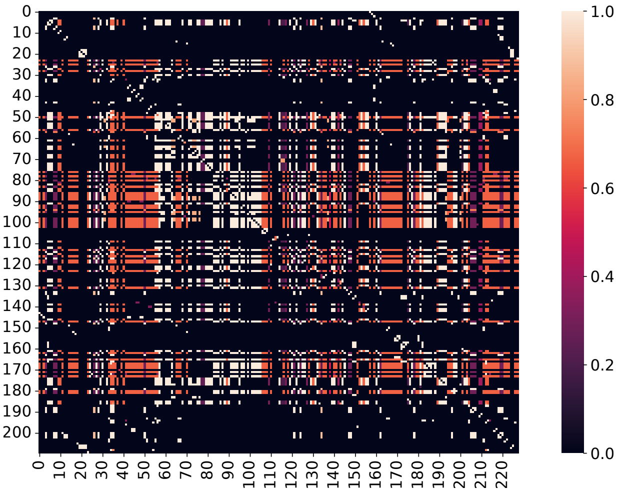


(b) Maximum Similarity Score.

Figure A.45: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptySubSeq`.



(a) Minimum Similarity Score.



(b) Maximum Similarity Score.

Figure A.46: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptySubSet`.

Appendix A.4.4. TrainTicket: Concrete Execution

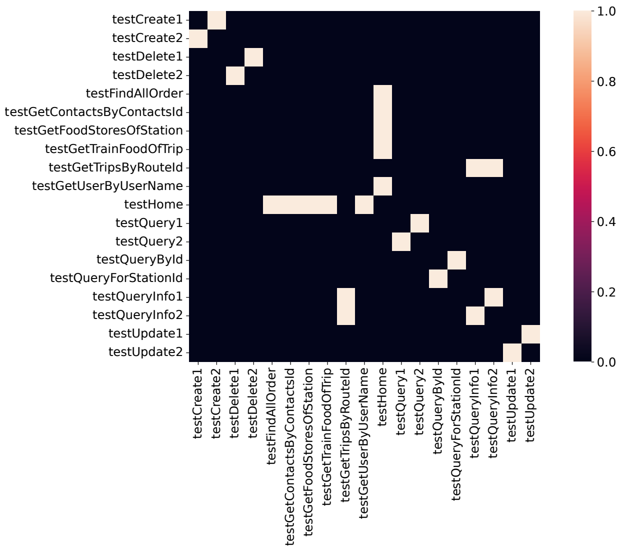


Figure A.47: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyCommonSeq`.

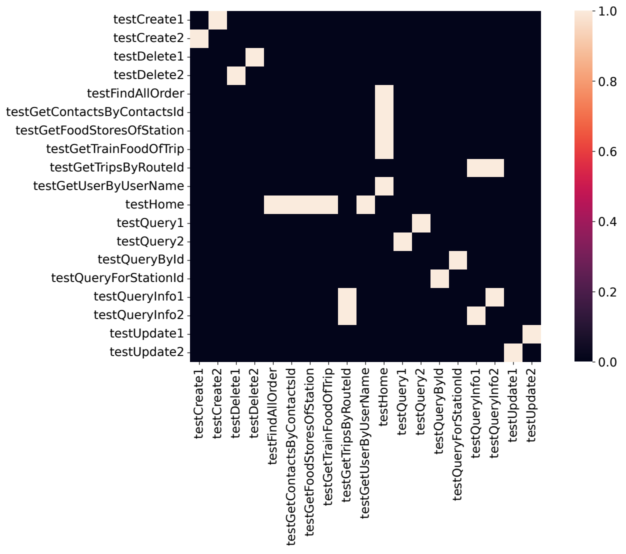


Figure A.48: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyEqSeq`.

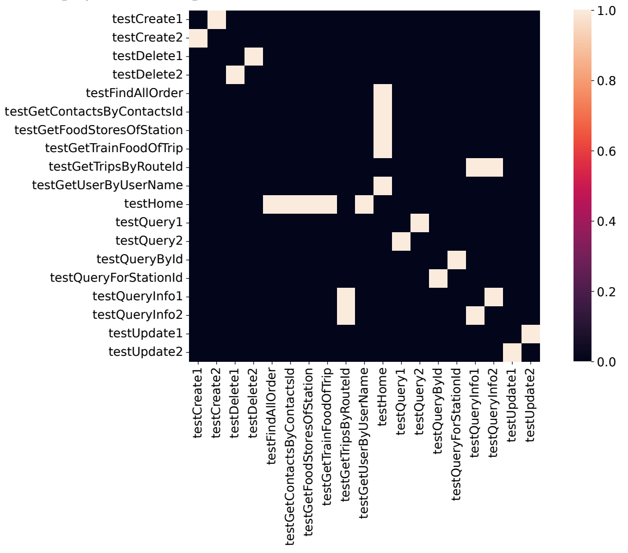


Figure A.49: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyEqSet`.

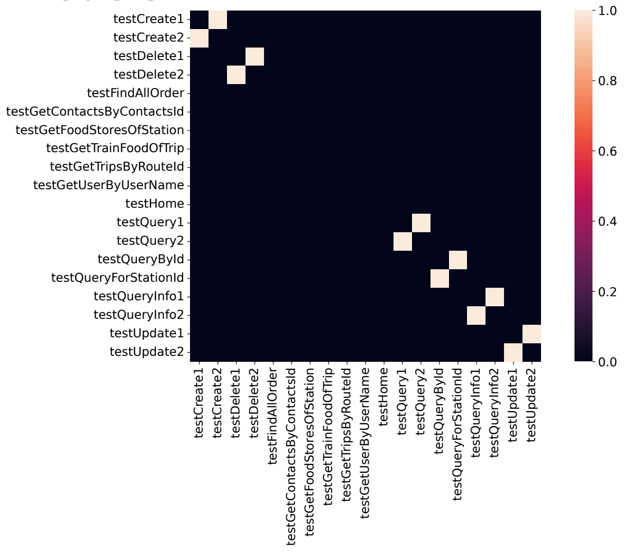
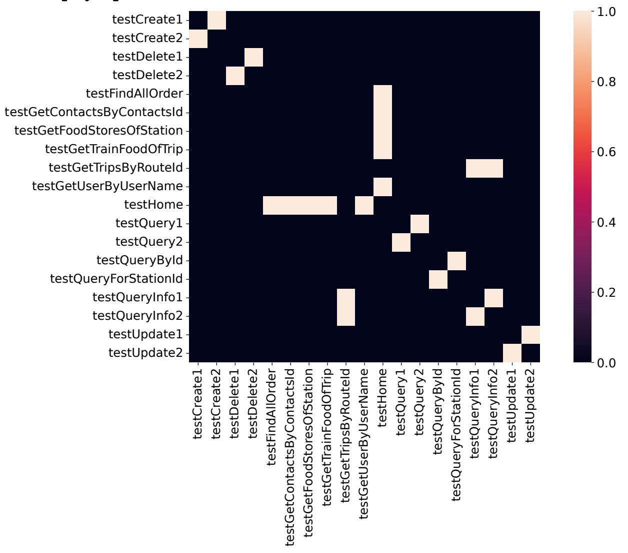


Figure A.50: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptyIntersection`.



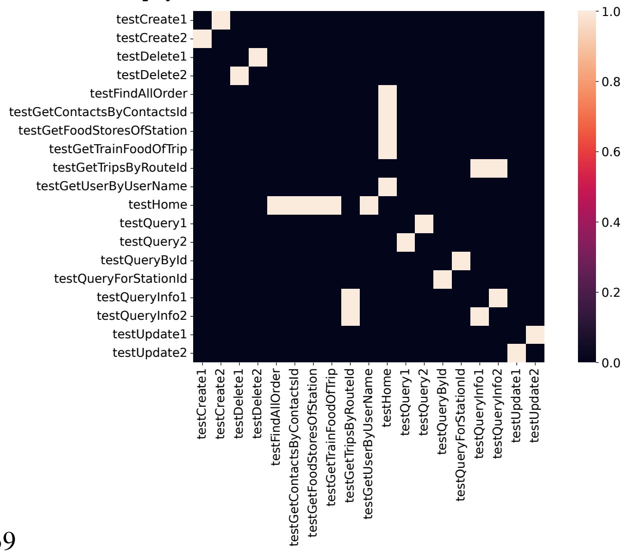


Figure A.51: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptySubSeq`.

Figure A.52: *Subject*: TrainTicket; *Domain*: `endpoint`; *Criterion*: `nonemptySubSet`.

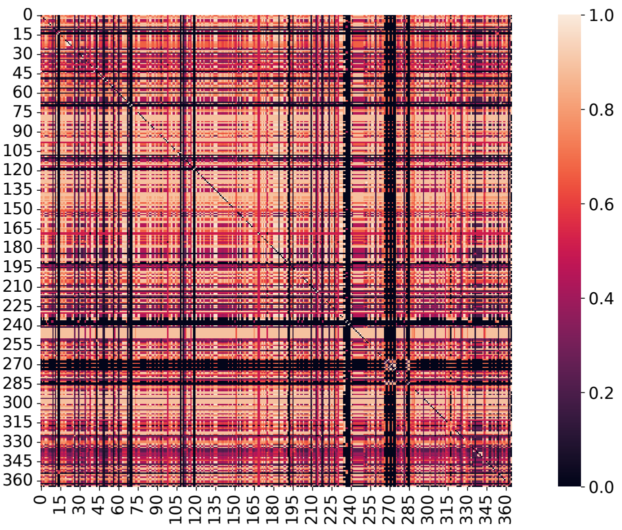


Figure A.53: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyCommonSeq`.

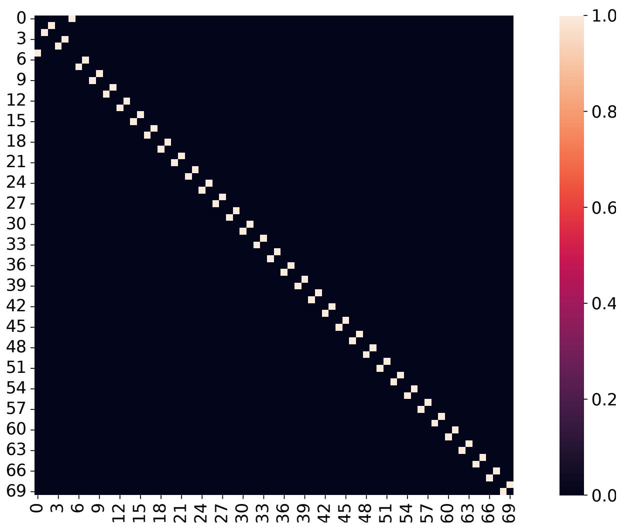


Figure A.54: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyEqSeq`.

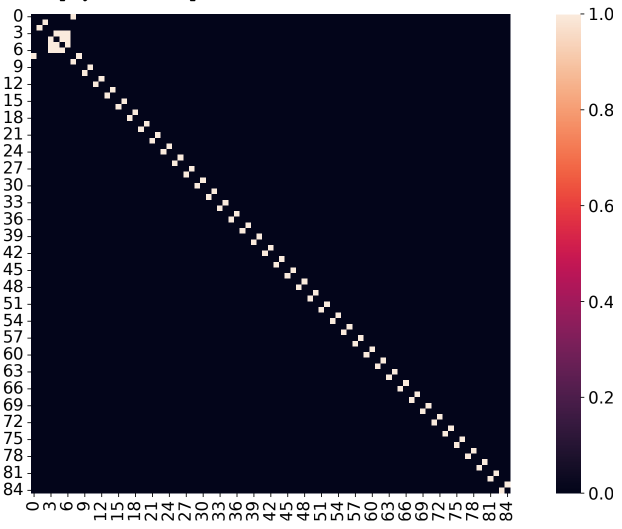


Figure A.55: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyEqSet`.

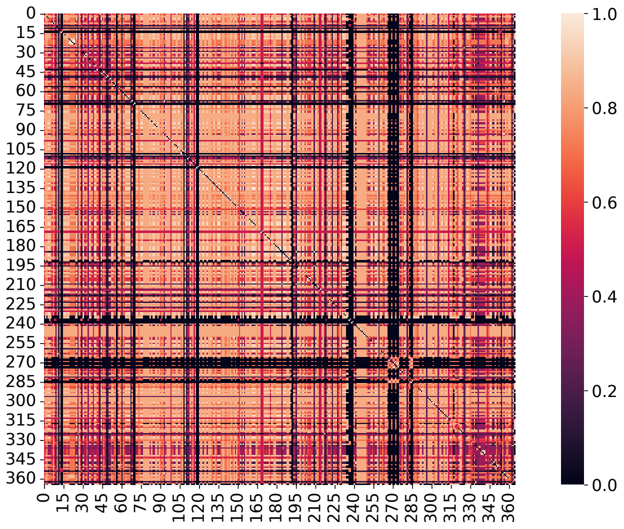


Figure A.56: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptyIntersection`.

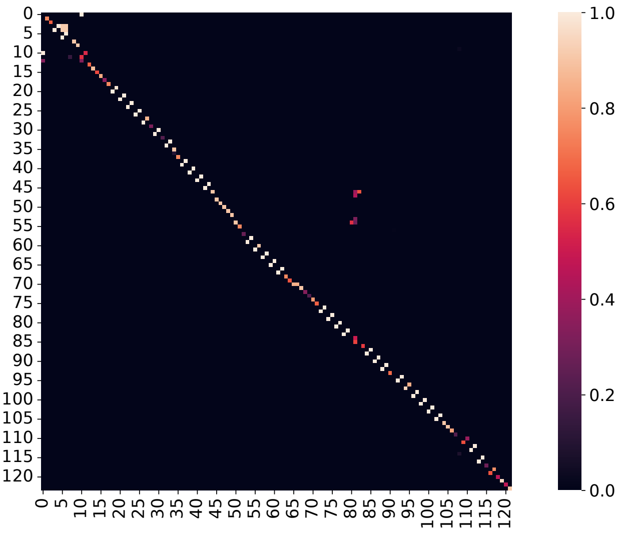


Figure A.57: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptySubSeq`.

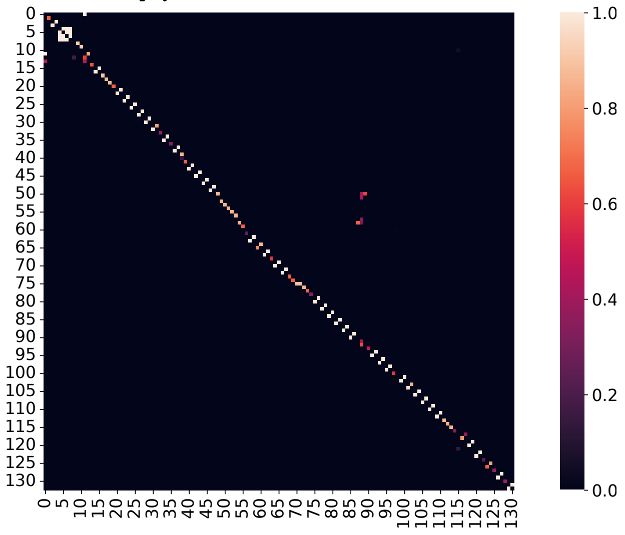


Figure A.58: *Subject*: TrainTicket; *Domain*: `invokes`; *Criterion*: `nonemptySubSet`.

Appendix A.5. Test Program Name Mapping for TrainTicket

Appendix A.5.1. Symbolic Execution

Table A.3: *Domain*: `invokes`; *Criterion*: `nonemptyCommonSeq`. Refers to Figures 16 and A.41.

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
1	testAddConfig	1	testAddConfig
2	testAddContact	2	testAddContact
3	testAddContacts	3	testAddContacts
4	testAddCreateNewOrder	4	testAddCreateNewOrder
5	testAddMoney	5	testAddMoney
6	testAddMoney1	6	testAddMoney1
7	testAddMoney2	7	testAddMoney2
8	testAddNewOrder1	8	testAddNewOrder1
9	testAddNewOrder2	9	testAddNewOrder2
10	testAddNewSecurityConfig1	10	testAddNewSecurityConfig1
11	testAddNewSecurityConfig2	11	testAddNewSecurityConfig2
12	testAddOrder	12	testAddOrder
13	testAddOrder1	13	testAddOrder1
14	testAddOrder2	14	testAddOrder2
15	testAddPrice	15	testAddPrice
16	testAddRoute	16	testAddRoute
17	testAddStation	17	testAddStation
18	testAddTrain	18	testAddTrain
19	testAddTravel	19	testAddTravel
20	testAddTravel1	20	testAddTravel1
21	testAddTravel2	21	testAddTravel2
22	testAddTravel3	22	testAddTravel3
23	testAddTravel4	23	testAddTravel4
24	testAddUser	24	testAddUser
25	testAdminQueryAll	25	testAdminQueryAll
26	testAdminQueryAll1	26	testAdminQueryAll1
27	testAlterOrder1	27	testAlterOrder1
28	testCancelOrder1	28	testCancelOrder1
29	testCancelOrder2	29	testCancelOrder2
30	testCheck	30	testCheck
31	testCheckSecurityAboutOrder	31	testCheckSecurityAboutOrder
32	testCheckStationExists	32	testCheckStationExists
33	testCollectTicket	33	testCollectTicket
34	testCreate	34	testCreate
35	testCreate1	35	testCreate1
36	testCreate2	36	testCreate2
37	testCreateAccount	37	testCreateAccount
38	testCreateAccount1	38	testCreateAccount1
39	testCreateAccount2	39	testCreateAccount2
40	testCreateAndModify1	40	testCreateAndModify1
41	testCreateAndModify2	41	testCreateAndModify2
42	testCreateAndModify3	42	testCreateAndModify3
43	testCreateAndModifyRoute	43	testCreateAndModifyRoute
44	testCreateConfig	44	testCreateConfig
45	testCreateContacts1	45	testCreateContacts1
46	testCreateContacts2	46	testCreateContacts2
47	testCreateFoodOrder	47	testCreateFoodOrder
48	testCreateFoodOrder1	48	testCreateFoodOrder1
49	testCreateFoodStore1	49	testCreateFoodStore1
50	testCreateFoodStore2	50	testCreateFoodStore2
51	testCreateNewAssurance	51	testCreateNewAssurance
52	testCreateNewContacts	52	testCreateNewContacts
53	testCreateNewContactsAdmin	53	testCreateNewContactsAdmin
54	testCreateNewOrder	54	testCreateNewOrder
55	testCreateNewPriceConfig1	55	testCreateNewPriceConfig1
56	testCreateTrainFood1	56	testCreateTrainFood1
57	testCreateTrainFood2	57	testCreateTrainFood2
58	testCreateTrip	58	testCreateTrip
59	testDelete	59	testDelete
60	testDelete1	60	testDelete1
61	testDelete2	61	testDelete2
62	testDeleteConfig	62	testDeleteConfig
63	testDeleteContact	63	testDeleteContact
64	testDeleteContacts	64	testDeleteContacts
65	testDeleteFoodOrder	65	testDeleteFoodOrder
66	testDeleteOrder	66	testDeleteOrder
67	testDeleteOrder1	67	testDeleteOrder1
68	testDeleteOrder2	68	testDeleteOrder2
69	testDeletePrice	69	testDeletePrice
70	testDeletePriceConfig1	70	testDeletePriceConfig1
71	testDeletePriceConfig2	71	testDeletePriceConfig2
72	testDeleteRoute	72	testDeleteRoute
73	testDeleteRoute1	73	testDeleteRoute1
74	testDeleteRoute2	74	testDeleteRoute2
75	testDeleteStation	75	testDeleteStation
76	testDeleteTrain	76	testDeleteTrain
77	testDeleteTravel	77	testDeleteTravel
78	testDeleteTravel1	78	testDeleteTravel1
79	testDeleteTravel2	79	testDeleteTravel2
80	testDeleteTrip	80	testDeleteTrip
81	testDeleteUser	81	testDeleteUser
82	testDipatchSeat	82	testDipatchSeat
83	testDistributeSeat1	83	testDistributeSeat1
84	testDistributeSeat2	84	testDistributeSeat2
85	testDrawBack	85	testDrawBack
86	testDrawBack1	86	testDrawBack1
87	testDrawBack2	87	testDrawBack2
88	testExecuteTicket	88	testExecuteTicket
89	testFindAllFoodOrder	89	testFindAllFoodOrder
90	testFindAllFoodOrder1	90	testFindAllFoodOrder1
91	testFindAllOrder	91	testFindAllOrder
92	testFindAllPriceConfig2	92	testFindAllPriceConfig2
93	testFindAllSecurityConfig	93	testFindAllSecurityConfig
94	testFindAllSecurityConfig1	94	testFindAllSecurityConfig1
95	testFindByConsignee	95	testFindByConsignee
96	testFindById	96	testFindById
97	testFindByRouteIdAndTrainType1	97	testFindByRouteIdAndTrainType1
98	testFindByRouteIdAndTrainType2	98	testFindByRouteIdAndTrainType2
99	testFindFoodOrderByOrderId	99	testFindFoodOrderByOrderId
100	testGetAccount	100	testGetAccount
101	testGetAllAssuranceType	101	testGetAllAssuranceType

	TP2		TP1
TP Number	TP Name	TP Number	TP Name
102	testGetAllAssuranceTypes	102	testGetAllAssuranceTypes
103	testGetAllAssurances	103	testGetAllAssurances
104	testGetAllAssurances1	104	testGetAllAssurances1
105	testGetAllConfigs	105	testGetAllConfigs
106	testGetAllContacts	106	testGetAllContacts
107	testGetAllContacts1	107	testGetAllContacts1
108	testGetAllFood	108	testGetAllFood
109	testGetAllFoodStores	109	testGetAllFoodStores
110	testGetAllOrders	110	testGetAllOrders
111	testGetAllOrders1	111	testGetAllOrders1
112	testGetAllOrders2	112	testGetAllOrders2
113	testGetAllPrices	113	testGetAllPrices
114	testGetAllRoutes	114	testGetAllRoutes
115	testGetAllRoutes1	115	testGetAllRoutes1
116	testGetAllStations	116	testGetAllStations
117	testGetAllTrains	117	testGetAllTrains
118	testGetAllTravels	118	testGetAllTravels
119	testGetAllTravels1	119	testGetAllTravels1
120	testGetAllTravels2	120	testGetAllTravels2
121	testGetAllUser	121	testGetAllUser
122	testGetAllUsers	122	testGetAllUsers
123	testGetByCheapest	123	testGetByCheapest
124	testGetByMinStation	124	testGetByMinStation
125	testGetByQuickest	125	testGetByQuickest
126	testGetCheapest	126	testGetCheapest
127	testGetCheapestRoutes	127	testGetCheapestRoutes
128	testGetFoodStoresByStationIds	128	testGetFoodStoresByStationIds
129	testGetFoodStoresByStationIds1	129	testGetFoodStoresByStationIds1
130	testGetFoodStoresByStationIds2	130	testGetFoodStoresByStationIds2
131	testGetFoodStoresOfStation	131	testGetFoodStoresOfStation
132	testGetLeftTicketOfInterva2	132	testGetLeftTicketOfInterva2
133	testGetLeftTicketOfInterval	133	testGetLeftTicketOfInterval
134	testGetMinStation	134	testGetMinStation
135	testGetMinStopStations	135	testGetMinStopStations
136	testGetOrderById	136	testGetOrderById
137	testGetOrderById1	137	testGetOrderById1
138	testGetOrderById2	138	testGetOrderById2
139	testGetOrderPrice	139	testGetOrderPrice
140	testGetOrderPrice1	140	testGetOrderPrice1
141	testGetOrderPrice2	141	testGetOrderPrice2
142	testGetPriceByWeightAndRegion	142	testGetPriceByWeightAndRegion
143	testGetPriceConfig	143	testGetPriceConfig
144	testGetPriceInfo	144	testGetPriceInfo
145	testGetQuickest	145	testGetQuickest
146	testGetQuickestRoutes	146	testGetQuickestRoutes
147	testGetRouteById1	147	testGetRouteById1
148	testGetRouteById2	148	testGetRouteById2
149	testGetRouteByTripId	149	testGetRouteByTripId
150	testGetRouteByTripId1	150	testGetRouteByTripId1
151	testGetRouteByTripId2	151	testGetRouteByTripId2
152	testGetSoldTickets1	152	testGetSoldTickets1
153	testGetSoldTickets2	153	testGetSoldTickets2
154	testGetTicketListByDateAndTripId	154	testGetTicketListByDateAndTripId
155	testGetToken1	155	testGetToken1
156	testGetToken2	156	testGetToken2
157	testGetTrainFoodOfTrip	157	testGetTrainFoodOfTrip
158	testGetTrainTypeByTripId	158	testGetTrainTypeByTripId
159	testGetTransferResult	159	testGetTransferResult
160	testGetTransferSearch	160	testGetTransferSearch
161	testGetTripAllDetailInfo	161	testGetTripAllDetailInfo
162	testGetTripByRoute2	162	testGetTripByRoute2
163	testGetTripsByRouteId	163	testGetTripsByRouteId
164	testHome	164	testHome
165	testInitOrder1	165	testInitOrder1
166	testInitOrder2	166	testInitOrder2
167	testInitPayment1	167	testInitPayment1
168	testInitPayment2	168	testInitPayment2
169	testInsertConsign	169	testInsertConsign
170	testListFoodStores1	170	testListFoodStores1
171	testListFoodStoresByStationId1	171	testListFoodStoresByStationId1
172	testListFoodStoresByStationId2	172	testListFoodStoresByStationId2
173	testListTrainFood1	173	testListTrainFood1
174	testListTrainFoodByTripId1	174	testListTrainFoodByTripId1
175	testListTrainFoodByTripId2	175	testListTrainFoodByTripId2
176	testModifyAssurance	176	testModifyAssurance
177	testModifyConfig	177	testModifyConfig
178	testModifyContact	178	testModifyContact
179	testModifyContacts	179	testModifyContacts
180	testModifyOrder	180	testModifyOrder
181	testModifyOrder1	181	testModifyOrder1
182	testModifyOrder2	182	testModifyOrder2
183	testModifyPrice	183	testModifyPrice
184	testModifyPriceConfig	184	testModifyPriceConfig
185	testModifySecurityConfig1	185	testModifySecurityConfig1
186	testModifySecurityConfig2	186	testModifySecurityConfig2
187	testModifyStation	187	testModifyStation
188	testModifyTrain	188	testModifyTrain
189	testOrderCancelSuccess	189	testOrderCancelSuccess
190	testOrderChangedSuccess	190	testOrderChangedSuccess
191	testOrderCreateSuccess	191	testOrderCreateSuccess
192	testPay	192	testPay
193	testPay1	193	testPay1
194	testPay2	194	testPay2
195	testPayDifference	195	testPayDifference
196	testPayOrder	196	testPayOrder
197	testPayOrder1	197	testPayOrder1
198	testPayOrder2	198	testPayOrder2
199	testPreserve	199	testPreserve
200	testPreserveSuccess	200	testPreserveSuccess
201	testQuery	201	testQuery
202	testQuery1	202	testQuery1
203	testQuery2	203	testQuery2
204	testQueryAccount	204	testQueryAccount
205	testQueryAddMoney	205	testQueryAddMoney
206	testQueryAddMoney1	206	testQueryAddMoney1
207	testQueryAll	207	testQueryAll
208	testQueryAll1	208	testQueryAll1
209	testQueryAlreadySoldOrders	209	testQueryAlreadySoldOrders
210	testQueryByConsignee1	210	testQueryByConsignee1
211	testQueryByConsignee2	211	testQueryByConsignee2

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
212	testQueryById	212	testQueryById
213	testQueryByStartAndTerminal	213	testQueryByStartAndTerminal
214	testQueryForStationId	214	testQueryForStationId
215	testQueryForTravel	215	testQueryForTravel
216	testQueryInfo1	216	testQueryInfo1
217	testQueryInfo2	217	testQueryInfo2
218	testQueryOrders	218	testQueryOrders
219	testQueryOrdersForRefresh	219	testQueryOrdersForRefresh
220	testQueryPayment	220	testQueryPayment
221	testQueryPayment1	221	testQueryPayment1
222	testQueryTrainType	222	testQueryTrainType
223	testRebook	223	testRebook
224	testRetrieve	224	testRetrieve
225	testRetrieve1	225	testRetrieve1
226	testRetrieve2	226	testRetrieve2
227	testSaveChanges1	227	testSaveChanges1
228	testSaveChanges2	228	testSaveChanges2
229	testSaveOrderInfo	229	testSaveOrderInfo
230	testSaveUser	230	testSaveUser
231	testSearchMinStopStations	231	testSearchMinStopStations
232	testSearchQuickestResult	232	testSearchQuickestResult
233	testSendEmail	233	testSendEmail
234	testTicketCollect1	234	testTicketCollect1
235	testTicketCollect2	235	testTicketCollect2
236	testTicketExecute1	236	testTicketExecute1
237	testTicketExecute2	237	testTicketExecute2
238	testUpdate	238	testUpdate
239	testUpdate1	239	testUpdate1
240	testUpdate2	240	testUpdate2
241	testUpdateConfig	241	testUpdateConfig
242	testUpdateConsign	242	testUpdateConsign
243	testUpdateFoodOrder	243	testUpdateFoodOrder
244	testUpdateFoodOrder1	244	testUpdateFoodOrder1
245	testUpdateOrder	245	testUpdateOrder
246	testUpdateOrder1	246	testUpdateOrder1
247	testUpdateOrder2	247	testUpdateOrder2
248	testUpdatePriceConfig1	248	testUpdatePriceConfig1
249	testUpdatePriceConfig2	249	testUpdatePriceConfig2
250	testUpdateTravel	250	testUpdateTravel
251	testUpdateTravel1	251	testUpdateTravel1
252	testUpdateTravel2	252	testUpdateTravel2
253	testUpdateTrip	253	testUpdateTrip
254	testUpdateUser	254	testUpdateUser

Table A.4: *Domain*: `invokes`; *Criterion*: `nonemptyEqSeq`. Refers to Figure A.42.

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
1	testAddConfig	1	testAddConfig
2	testAddContact	2	testAddContact
3	testAddContacts	3	testAddContacts
4	testAddCreateNewOrder	4	testAddCreateNewOrder
5	testAddMoney	5	testAddMoney
6	testAddMoney1	6	testAddMoney1
7	testAddMoney2	7	testAddMoney2
8	testAddNewOrder1	8	testAddNewOrder1
9	testAddNewOrder2	9	testAddNewOrder2
10	testAddNewSecurityConfig1	10	testAddNewSecurityConfig1
11	testAddNewSecurityConfig2	11	testAddNewSecurityConfig2
12	testAddOrder	12	testAddOrder
13	testAddOrder1	13	testAddOrder1
14	testAddOrder2	14	testAddOrder2
15	testAddPrice	15	testAddPrice
16	testAddStation	16	testAddStation
17	testAddTrain	17	testAddTrain
18	testAddTravel	18	testAddTravel
19	testAddTravel1	19	testAddTravel1
20	testAddTravel2	20	testAddTravel2
21	testAddTravel3	21	testAddTravel3
22	testAddTravel4	22	testAddTravel4
23	testAddUser	23	testAddUser
24	testAdminQueryAll	24	testAdminQueryAll
25	testAdminQueryAll1	25	testAdminQueryAll1
26	testCancelOrder1	26	testCancelOrder1
27	testCancelOrder2	27	testCancelOrder2
28	testCheck	28	testCheck
29	testCheckSecurityAboutOrder	29	testCheckSecurityAboutOrder
30	testCollectTicket	30	testCollectTicket
31	testCreate	31	testCreate
32	testCreate1	32	testCreate1
33	testCreate2	33	testCreate2
34	testCreateConfig	34	testCreateConfig
35	testCreateContacts1	35	testCreateContacts1
36	testCreateContacts2	36	testCreateContacts2
37	testCreateFoodOrder	37	testCreateFoodOrder
38	testCreateFoodOrder1	38	testCreateFoodOrder1
39	testCreateFoodStore1	39	testCreateFoodStore1
40	testCreateFoodStore2	40	testCreateFoodStore2
41	testCreateNewContacts	41	testCreateNewContacts
42	testCreateNewContactsAdmin	42	testCreateNewContactsAdmin
43	testCreateNewOrder	43	testCreateNewOrder
44	testCreateNewPriceConfig1	44	testCreateNewPriceConfig1
45	testCreateTrainFood1	45	testCreateTrainFood1
46	testCreateTrainFood2	46	testCreateTrainFood2
47	testCreateTrip	47	testCreateTrip
48	testDelete	48	testDelete
49	testDelete1	49	testDelete1
50	testDelete2	50	testDelete2
51	testDeleteConfig	51	testDeleteConfig
52	testDeleteContact	52	testDeleteContact
53	testDeleteContacts	53	testDeleteContacts
54	testDeleteFoodOrder	54	testDeleteFoodOrder
55	testDeleteOrder	55	testDeleteOrder
56	testDeleteOrder1	56	testDeleteOrder1
57	testDeleteOrder2	57	testDeleteOrder2

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
58	testDeletePrice	58	testDeletePrice
59	testDeletePriceConfig1	59	testDeletePriceConfig1
60	testDeletePriceConfig2	60	testDeletePriceConfig2
61	testDeleteRoute	61	testDeleteRoute
62	testDeleteStation	62	testDeleteStation
63	testDeleteTrain	63	testDeleteTrain
64	testDeleteTravel	64	testDeleteTravel
65	testDeleteTravel1	65	testDeleteTravel1
66	testDeleteTravel2	66	testDeleteTravel2
67	testDeleteTrip	67	testDeleteTrip
68	testDeleteUser	68	testDeleteUser
69	testDistributeSeat1	69	testDistributeSeat1
70	testDistributeSeat2	70	testDistributeSeat2
71	testDrawBack	71	testDrawBack
72	testDrawBack1	72	testDrawBack1
73	testDrawBack2	73	testDrawBack2
74	testExecuteTicket	74	testExecuteTicket
75	testFindAllFoodOrder	75	testFindAllFoodOrder
76	testFindAllOrder	76	testFindAllOrder
77	testFindAllSecurityConfig	77	testFindAllSecurityConfig
78	testFindByConsignee	78	testFindByConsignee
79	testFindById	79	testFindById
80	testFindByRouteIdAndTrainType1	80	testFindByRouteIdAndTrainType1
81	testFindFoodOrderByOrderId	81	testFindFoodOrderByOrderId
82	testGetAllAssuranceType	82	testGetAllAssuranceType
83	testGetAllAssurances	83	testGetAllAssurances
84	testGetAllConfigs	84	testGetAllConfigs
85	testGetAllContacts	85	testGetAllContacts
86	testGetAllFood	86	testGetAllFood
87	testGetAllFoodStores	87	testGetAllFoodStores
88	testGetAllOrders	88	testGetAllOrders
89	testGetAllOrders2	89	testGetAllOrders2
90	testGetAllPrices	90	testGetAllPrices
91	testGetAllRoutes	91	testGetAllRoutes
92	testGetAllStations	92	testGetAllStations
93	testGetAllTrains	93	testGetAllTrains
94	testGetAllTravels	94	testGetAllTravels
95	testGetAllUser	95	testGetAllUser
96	testGetAllUsers	96	testGetAllUsers
97	testGetByCheapest	97	testGetByCheapest
98	testGetByMinStation	98	testGetByMinStation
99	testGetByQuickest	99	testGetByQuickest
100	testGetCheapestRoutes	100	testGetCheapestRoutes
101	testGetFoodStoresByStationIds	101	testGetFoodStoresByStationIds
102	testGetFoodStoresByStationIds2	102	testGetFoodStoresByStationIds2
103	testGetFoodStoresOfStation	103	testGetFoodStoresOfStation
104	testGetLeftTicketOfInterval	104	testGetLeftTicketOfInterval
105	testGetMinStopStations	105	testGetMinStopStations
106	testGetOrderById	106	testGetOrderById
107	testGetOrderById1	107	testGetOrderById1
108	testGetOrderById2	108	testGetOrderById2
109	testGetOrderPrice	109	testGetOrderPrice
110	testGetOrderPrice1	110	testGetOrderPrice1
111	testGetOrderPrice2	111	testGetOrderPrice2
112	testGetPriceConfig	112	testGetPriceConfig
113	testGetPriceInfo	113	testGetPriceInfo
114	testGetQuickestRoutes	114	testGetQuickestRoutes
115	testGetRouteById1	115	testGetRouteById1
116	testGetRouteByTripId	116	testGetRouteByTripId
117	testGetRouteByTripId1	117	testGetRouteByTripId1
118	testGetRouteByTripId2	118	testGetRouteByTripId2
119	testGetSoldTickets2	119	testGetSoldTickets2
120	testGetTicketListByDateAndTripId	120	testGetTicketListByDateAndTripId
121	testGetTrainFoodOfTrip	121	testGetTrainFoodOfTrip
122	testGetTrainTypeByTripId	122	testGetTrainTypeByTripId
123	testGetTripByRoute2	123	testGetTripByRoute2
124	testGetTripsByRouteId	124	testGetTripsByRouteId
125	testHome	125	testHome
126	testInitOrder1	126	testInitOrder1
127	testInitOrder2	127	testInitOrder2
128	testInitPayment1	128	testInitPayment1
129	testInitPayment2	129	testInitPayment2
130	testInsertConsign	130	testInsertConsign
131	testListFoodStoresByStationId2	131	testListFoodStoresByStationId2
132	testListTrainFoodByTripId2	132	testListTrainFoodByTripId2
133	testModifyAssurance	133	testModifyAssurance
134	testModifyConfig	134	testModifyConfig
135	testModifyContact	135	testModifyContact
136	testModifyContacts	136	testModifyContacts
137	testModifyOrder	137	testModifyOrder
138	testModifyOrder1	138	testModifyOrder1
139	testModifyOrder2	139	testModifyOrder2
140	testModifyPrice	140	testModifyPrice
141	testModifySecurityConfig1	141	testModifySecurityConfig1
142	testModifySecurityConfig2	142	testModifySecurityConfig2
143	testModifyStation	143	testModifyStation
144	testModifyTrain	144	testModifyTrain
145	testOrderCancelSuccess	145	testOrderCancelSuccess
146	testOrderChangedSuccess	146	testOrderChangedSuccess
147	testOrderCreateSuccess	147	testOrderCreateSuccess
148	testPay	148	testPay
149	testPay1	149	testPay1
150	testPay2	150	testPay2
151	testPayDifference	151	testPayDifference
152	testPayOrder	152	testPayOrder
153	testPayOrder1	153	testPayOrder1
154	testPayOrder2	154	testPayOrder2
155	testPreserveSuccess	155	testPreserveSuccess
156	testQuery	156	testQuery
157	testQuery1	157	testQuery1
158	testQuery2	158	testQuery2
159	testQueryAccount	159	testQueryAccount
160	testQueryAddMoney	160	testQueryAddMoney
161	testQueryAll	161	testQueryAll
162	testQueryAll1	162	testQueryAll1
163	testQueryAlreadySoldOrders	163	testQueryAlreadySoldOrders
164	testQueryByConsignee2	164	testQueryByConsignee2
165	testQueryById	165	testQueryById
166	testQueryByStartAndTerminal	166	testQueryByStartAndTerminal
167	testQueryForStationId	167	testQueryForStationId

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
168	testQueryOrders	168	testQueryOrders
169	testQueryOrdersForRefresh	169	testQueryOrdersForRefresh
170	testQueryPayment	170	testQueryPayment
171	testQueryPayment1	171	testQueryPayment1
172	testQueryTrainType	172	testQueryTrainType
173	testRebook	173	testRebook
174	testRetrieve	174	testRetrieve
175	testRetrieve1	175	testRetrieve1
176	testRetrieve2	176	testRetrieve2
177	testSaveChanges1	177	testSaveChanges1
178	testSaveChanges2	178	testSaveChanges2
179	testSaveOrderInfo	179	testSaveOrderInfo
180	testUpdate	180	testUpdate
181	testUpdate1	181	testUpdate1
182	testUpdate2	182	testUpdate2
183	testUpdateConfig	183	testUpdateConfig
184	testUpdateConsign	184	testUpdateConsign
185	testUpdateFoodOrder	185	testUpdateFoodOrder
186	testUpdateFoodOrder1	186	testUpdateFoodOrder1
187	testUpdateOrder	187	testUpdateOrder
188	testUpdateOrder1	188	testUpdateOrder1
189	testUpdateOrder2	189	testUpdateOrder2
190	testUpdatePriceConfig1	190	testUpdatePriceConfig1
191	testUpdatePriceConfig2	191	testUpdatePriceConfig2
192	testUpdateTravel	192	testUpdateTravel
193	testUpdateTravel1	193	testUpdateTravel1
194	testUpdateTravel2	194	testUpdateTravel2
195	testUpdateTrip	195	testUpdateTrip
196	testUpdateUser	196	testUpdateUser

Table A.5: *Domain*: `invokes`; *Criterion*: `nonemptyEqSet`. Refers to Figure A.43.

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
1	testAddConfig	1	testAddConfig
2	testAddContact	2	testAddContact
3	testAddContacts	3	testAddContacts
4	testAddCreateNewOrder	4	testAddCreateNewOrder
5	testAddMoney	5	testAddMoney
6	testAddMoney1	6	testAddMoney1
7	testAddMoney2	7	testAddMoney2
8	testAddNewOrder1	8	testAddNewOrder1
9	testAddNewOrder2	9	testAddNewOrder2
10	testAddNewSecurityConfig1	10	testAddNewSecurityConfig1
11	testAddNewSecurityConfig2	11	testAddNewSecurityConfig2
12	testAddOrder	12	testAddOrder
13	testAddOrder1	13	testAddOrder1
14	testAddOrder2	14	testAddOrder2
15	testAddPrice	15	testAddPrice
16	testAddStation	16	testAddStation
17	testAddTrain	17	testAddTrain
18	testAddTravel	18	testAddTravel
19	testAddTravel1	19	testAddTravel1
20	testAddTravel2	20	testAddTravel2
21	testAddTravel3	21	testAddTravel3
22	testAddTravel4	22	testAddTravel4
23	testAddUser	23	testAddUser
24	testAdminQueryAll	24	testAdminQueryAll
25	testAdminQueryAll1	25	testAdminQueryAll1
26	testCancelOrder1	26	testCancelOrder1
27	testCancelOrder2	27	testCancelOrder2
28	testCheck	28	testCheck
29	testCheckSecurityAboutOrder	29	testCheckSecurityAboutOrder
30	testCollectTicket	30	testCollectTicket
31	testCreate	31	testCreate
32	testCreate1	32	testCreate1
33	testCreate2	33	testCreate2
34	testCreateConfig	34	testCreateConfig
35	testCreateContacts1	35	testCreateContacts1
36	testCreateContacts2	36	testCreateContacts2
37	testCreateFoodOrder	37	testCreateFoodOrder
38	testCreateFoodOrder1	38	testCreateFoodOrder1
39	testCreateFoodStore1	39	testCreateFoodStore1
40	testCreateFoodStore2	40	testCreateFoodStore2
41	testCreateNewContacts	41	testCreateNewContacts
42	testCreateNewContactsAdmin	42	testCreateNewContactsAdmin
43	testCreateNewOrder	43	testCreateNewOrder
44	testCreateNewPriceConfig1	44	testCreateNewPriceConfig1
45	testCreateTrainFood1	45	testCreateTrainFood1
46	testCreateTrainFood2	46	testCreateTrainFood2
47	testCreateTrip	47	testCreateTrip
48	testDelete	48	testDelete
49	testDelete1	49	testDelete1
50	testDelete2	50	testDelete2
51	testDeleteConfig	51	testDeleteConfig
52	testDeleteContact	52	testDeleteContact
53	testDeleteContacts	53	testDeleteContacts
54	testDeleteFoodOrder	54	testDeleteFoodOrder
55	testDeleteOrder	55	testDeleteOrder
56	testDeleteOrder1	56	testDeleteOrder1
57	testDeleteOrder2	57	testDeleteOrder2
58	testDeletePrice	58	testDeletePrice
59	testDeletePriceConfig1	59	testDeletePriceConfig1
60	testDeletePriceConfig2	60	testDeletePriceConfig2
61	testDeleteRoute	61	testDeleteRoute
62	testDeleteStation	62	testDeleteStation
63	testDeleteTrain	63	testDeleteTrain
64	testDeleteTravel	64	testDeleteTravel
65	testDeleteTravel1	65	testDeleteTravel1
66	testDeleteTravel2	66	testDeleteTravel2
67	testDeleteTrip	67	testDeleteTrip
68	testDeleteUser	68	testDeleteUser
69	testDistributeSeat1	69	testDistributeSeat1
70	testDistributeSeat2	70	testDistributeSeat2
71	testDrawBack	71	testDrawBack

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
72	testDrawBack1	72	testDrawBack1
73	testDrawBack2	73	testDrawBack2
74	testExecuteTicket	74	testExecuteTicket
75	testFindAllFoodOrder	75	testFindAllFoodOrder
76	testFindAllOrder	76	testFindAllOrder
77	testFindAllSecurityConfig	77	testFindAllSecurityConfig
78	testFindByConsignee	78	testFindByConsignee
79	testFindById	79	testFindById
80	testFindByRouteIdAndTrainType1	80	testFindByRouteIdAndTrainType1
81	testFindFoodOrderByOrderId	81	testFindFoodOrderByOrderId
82	testGetAllAssuranceType	82	testGetAllAssuranceType
83	testGetAllAssurances	83	testGetAllAssurances
84	testGetAllConfigs	84	testGetAllConfigs
85	testGetAllContacts	85	testGetAllContacts
86	testGetAllFood	86	testGetAllFood
87	testGetAllFoodStores	87	testGetAllFoodStores
88	testGetAllOrders	88	testGetAllOrders
89	testGetAllOrders2	89	testGetAllOrders2
90	testGetAllPrices	90	testGetAllPrices
91	testGetAllRoutes	91	testGetAllRoutes
92	testGetAllStations	92	testGetAllStations
93	testGetAllTrains	93	testGetAllTrains
94	testGetAllTravels	94	testGetAllTravels
95	testGetAllUser	95	testGetAllUser
96	testGetAllUsers	96	testGetAllUsers
97	testGetByCheapest	97	testGetByCheapest
98	testGetByMinStation	98	testGetByMinStation
99	testGetByQuickest	99	testGetByQuickest
100	testGetCheapestRoutes	100	testGetCheapestRoutes
101	testGetFoodStoresByStationIds	101	testGetFoodStoresByStationIds
102	testGetFoodStoresByStationIds2	102	testGetFoodStoresByStationIds2
103	testGetFoodStoresOfStation	103	testGetFoodStoresOfStation
104	testGetLeftTicketOfInterval	104	testGetLeftTicketOfInterval
105	testGetMinStopStations	105	testGetMinStopStations
106	testGetOrderById	106	testGetOrderById
107	testGetOrderById1	107	testGetOrderById1
108	testGetOrderById2	108	testGetOrderById2
109	testGetOrderPrice	109	testGetOrderPrice
110	testGetOrderPrice1	110	testGetOrderPrice1
111	testGetOrderPrice2	111	testGetOrderPrice2
112	testGetPriceConfig	112	testGetPriceConfig
113	testGetPriceInfo	113	testGetPriceInfo
114	testGetQuickestRoutes	114	testGetQuickestRoutes
115	testGetRouteById1	115	testGetRouteById1
116	testGetRouteByTripId	116	testGetRouteByTripId
117	testGetRouteByTripId1	117	testGetRouteByTripId1
118	testGetRouteByTripId2	118	testGetRouteByTripId2
119	testGetSoldTickets2	119	testGetSoldTickets2
120	testGetTicketListByDateAndTripId	120	testGetTicketListByDateAndTripId
121	testGetTrainFoodOfTrip	121	testGetTrainFoodOfTrip
122	testGetTrainTypeByTripId	122	testGetTrainTypeByTripId
123	testGetTripByRoute2	123	testGetTripByRoute2
124	testGetTripsByRouteId	124	testGetTripsByRouteId
125	testHome	125	testHome
126	testInitOrder1	126	testInitOrder1
127	testInitOrder2	127	testInitOrder2
128	testInitPayment1	128	testInitPayment1
129	testInitPayment2	129	testInitPayment2
130	testInsertConsign	130	testInsertConsign
131	testListFoodStoresByStationId2	131	testListFoodStoresByStationId2
132	testListTrainFoodByTripId2	132	testListTrainFoodByTripId2
133	testModifyAssurance	133	testModifyAssurance
134	testModifyConfig	134	testModifyConfig
135	testModifyContact	135	testModifyContact
136	testModifyContacts	136	testModifyContacts
137	testModifyOrder	137	testModifyOrder
138	testModifyOrder1	138	testModifyOrder1
139	testModifyOrder2	139	testModifyOrder2
140	testModifyPrice	140	testModifyPrice
141	testModifySecurityConfig1	141	testModifySecurityConfig1
142	testModifySecurityConfig2	142	testModifySecurityConfig2
143	testModifyStation	143	testModifyStation
144	testModifyTrain	144	testModifyTrain
145	testOrderCancelSuccess	145	testOrderCancelSuccess
146	testOrderChangedSuccess	146	testOrderChangedSuccess
147	testOrderCreateSuccess	147	testOrderCreateSuccess
148	testPay	148	testPay
149	testPay1	149	testPay1
150	testPay2	150	testPay2
151	testPayDifference	151	testPayDifference
152	testPayOrder	152	testPayOrder
153	testPayOrder1	153	testPayOrder1
154	testPayOrder2	154	testPayOrder2
155	testPreserveSuccess	155	testPreserveSuccess
156	testQuery	156	testQuery
157	testQuery1	157	testQuery1
158	testQuery2	158	testQuery2
159	testQueryAccount	159	testQueryAccount
160	testQueryAddMoney	160	testQueryAddMoney
161	testQueryAll	161	testQueryAll
162	testQueryAll1	162	testQueryAll1
163	testQueryAlreadySoldOrders	163	testQueryAlreadySoldOrders
164	testQueryByConsignee2	164	testQueryByConsignee2
165	testQueryById	165	testQueryById
166	testQueryByStartAndTerminal	166	testQueryByStartAndTerminal
167	testQueryForStationId	167	testQueryForStationId
168	testQueryOrders	168	testQueryOrders
169	testQueryOrdersForRefresh	169	testQueryOrdersForRefresh
170	testQueryPayment	170	testQueryPayment
171	testQueryPayment1	171	testQueryPayment1
172	testQueryTrainType	172	testQueryTrainType
173	testRebook	173	testRebook
174	testRetrieve	174	testRetrieve
175	testRetrieve1	175	testRetrieve1
176	testRetrieve2	176	testRetrieve2
177	testSaveChanges1	177	testSaveChanges1
178	testSaveChanges2	178	testSaveChanges2
179	testSaveOrderInfo	179	testSaveOrderInfo
180	testSearchMinStopStations	180	testSearchMinStopStations
181	testSearchQuickestResult	181	testSearchQuickestResult

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
182	testUpdate	182	testUpdate
183	testUpdate1	183	testUpdate1
184	testUpdate2	184	testUpdate2
185	testUpdateConfig	185	testUpdateConfig
186	testUpdateConsign	186	testUpdateConsign
187	testUpdateFoodOrder	187	testUpdateFoodOrder
188	testUpdateFoodOrder1	188	testUpdateFoodOrder1
189	testUpdateOrder	189	testUpdateOrder
190	testUpdateOrder1	190	testUpdateOrder1
191	testUpdateOrder2	191	testUpdateOrder2
192	testUpdatePriceConfig1	192	testUpdatePriceConfig1
193	testUpdatePriceConfig2	193	testUpdatePriceConfig2
194	testUpdateTravel	194	testUpdateTravel
195	testUpdateTravel1	195	testUpdateTravel1
196	testUpdateTravel2	196	testUpdateTravel2
197	testUpdateTrip	197	testUpdateTrip
198	testUpdateUser	198	testUpdateUser

Table A.6: *Domain*: `invokes`; *Criterion*: `nonemptyIntersection`. Refers to Figures 12b and A.44.

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
1	testAddConfig	1	testAddConfig
2	testAddContact	2	testAddContact
3	testAddContacts	3	testAddContacts
4	testAddCreateNewOrder	4	testAddCreateNewOrder
5	testAddMoney	5	testAddMoney
6	testAddMoney1	6	testAddMoney1
7	testAddMoney2	7	testAddMoney2
8	testAddNewOrder1	8	testAddNewOrder1
9	testAddNewOrder2	9	testAddNewOrder2
10	testAddNewSecurityConfig1	10	testAddNewSecurityConfig1
11	testAddNewSecurityConfig2	11	testAddNewSecurityConfig2
12	testAddOrder	12	testAddOrder
13	testAddOrder1	13	testAddOrder1
14	testAddOrder2	14	testAddOrder2
15	testAddPrice	15	testAddPrice
16	testAddRoute	16	testAddRoute
17	testAddStation	17	testAddStation
18	testAddTrain	18	testAddTrain
19	testAddTravel	19	testAddTravel
20	testAddTravel1	20	testAddTravel1
21	testAddTravel2	21	testAddTravel2
22	testAddTravel3	22	testAddTravel3
23	testAddTravel4	23	testAddTravel4
24	testAddUser	24	testAddUser
25	testAdminQueryAll	25	testAdminQueryAll
26	testAdminQueryAll1	26	testAdminQueryAll1
27	testAlterOrder1	27	testAlterOrder1
28	testCancelOrder1	28	testCancelOrder1
29	testCancelOrder2	29	testCancelOrder2
30	testCheck	30	testCheck
31	testCheckSecurityAboutOrder	31	testCheckSecurityAboutOrder
32	testCheckStationExists	32	testCheckStationExists
33	testCollectTicket	33	testCollectTicket
34	testCreate	34	testCreate
35	testCreate1	35	testCreate1
36	testCreate2	36	testCreate2
37	testCreateAccount	37	testCreateAccount
38	testCreateAccount1	38	testCreateAccount1
39	testCreateAccount2	39	testCreateAccount2
40	testCreateAndModify1	40	testCreateAndModify1
41	testCreateAndModify2	41	testCreateAndModify2
42	testCreateAndModify3	42	testCreateAndModify3
43	testCreateAndModifyRoute	43	testCreateAndModifyRoute
44	testCreateConfig	44	testCreateConfig
45	testCreateContacts1	45	testCreateContacts1
46	testCreateContacts2	46	testCreateContacts2
47	testCreateFoodOrder	47	testCreateFoodOrder
48	testCreateFoodOrder1	48	testCreateFoodOrder1
49	testCreateFoodStore1	49	testCreateFoodStore1
50	testCreateFoodStore2	50	testCreateFoodStore2
51	testCreateNewAssurance	51	testCreateNewAssurance
52	testCreateNewContacts	52	testCreateNewContacts
53	testCreateNewContactsAdmin	53	testCreateNewContactsAdmin
54	testCreateNewOrder	54	testCreateNewOrder
55	testCreateNewPriceConfig1	55	testCreateNewPriceConfig1
56	testCreateTrainFood1	56	testCreateTrainFood1
57	testCreateTrainFood2	57	testCreateTrainFood2
58	testCreateTrip	58	testCreateTrip
59	testDelete	59	testDelete
60	testDelete1	60	testDelete1
61	testDelete2	61	testDelete2
62	testDeleteConfig	62	testDeleteConfig
63	testDeleteContact	63	testDeleteContact
64	testDeleteContacts	64	testDeleteContacts
65	testDeleteFoodOrder	65	testDeleteFoodOrder
66	testDeleteOrder	66	testDeleteOrder
67	testDeleteOrder1	67	testDeleteOrder1
68	testDeleteOrder2	68	testDeleteOrder2
69	testDeletePrice	69	testDeletePrice
70	testDeletePriceConfig1	70	testDeletePriceConfig1
71	testDeletePriceConfig2	71	testDeletePriceConfig2
72	testDeleteRoute	72	testDeleteRoute
73	testDeleteRoute1	73	testDeleteRoute1
74	testDeleteRoute2	74	testDeleteRoute2
75	testDeleteStation	75	testDeleteStation
76	testDeleteTrain	76	testDeleteTrain
77	testDeleteTravel	77	testDeleteTravel
78	testDeleteTravel1	78	testDeleteTravel1
79	testDeleteTravel2	79	testDeleteTravel2
80	testDeleteTrip	80	testDeleteTrip
81	testDeleteUser	81	testDeleteUser
82	testDipatchSeat	82	testDipatchSeat
83	testDistributeSeat1	83	testDistributeSeat1

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
84	testDistributeSeat2	84	testDistributeSeat2
85	testDrawBack	85	testDrawBack
86	testDrawBack1	86	testDrawBack1
87	testDrawBack2	87	testDrawBack2
88	testExecuteTicket	88	testExecuteTicket
89	testFindAllFoodOrder	89	testFindAllFoodOrder
90	testFindAllFoodOrder1	90	testFindAllFoodOrder1
91	testFindAllOrder	91	testFindAllOrder
92	testFindAllPriceConfig2	92	testFindAllPriceConfig2
93	testFindAllSecurityConfig	93	testFindAllSecurityConfig
94	testFindAllSecurityConfig1	94	testFindAllSecurityConfig1
95	testFindByConsignee	95	testFindByConsignee
96	testFindById	96	testFindById
97	testFindByRouteIdAndTrainType1	97	testFindByRouteIdAndTrainType1
98	testFindByRouteIdAndTrainType2	98	testFindByRouteIdAndTrainType2
99	testFindFoodOrderByOrderId	99	testFindFoodOrderByOrderId
100	testGetAccount	100	testGetAccount
101	testGetAllAssuranceType	101	testGetAllAssuranceType
102	testGetAllAssuranceTypes	102	testGetAllAssuranceTypes
103	testGetAllAssurances	103	testGetAllAssurances
104	testGetAllAssurances1	104	testGetAllAssurances1
105	testGetAllConfigs	105	testGetAllConfigs
106	testGetAllContacts	106	testGetAllContacts
107	testGetAllContacts1	107	testGetAllContacts1
108	testGetAllFood	108	testGetAllFood
109	testGetAllFoodStores	109	testGetAllFoodStores
110	testGetAllOrders	110	testGetAllOrders
111	testGetAllOrders1	111	testGetAllOrders1
112	testGetAllOrders2	112	testGetAllOrders2
113	testGetAllPrices	113	testGetAllPrices
114	testGetAllRoutes	114	testGetAllRoutes
115	testGetAllRoutes1	115	testGetAllRoutes1
116	testGetAllStations	116	testGetAllStations
117	testGetAllTrains	117	testGetAllTrains
118	testGetAllTravels	118	testGetAllTravels
119	testGetAllTravels1	119	testGetAllTravels1
120	testGetAllTravels2	120	testGetAllTravels2
121	testGetAllUser	121	testGetAllUser
122	testGetAllUsers	122	testGetAllUsers
123	testGetByCheapest	123	testGetByCheapest
124	testGetByMinStation	124	testGetByMinStation
125	testGetByQuickest	125	testGetByQuickest
126	testGetCheapest	126	testGetCheapest
127	testGetCheapestRoutes	127	testGetCheapestRoutes
128	testGetFoodStoresByStationIds	128	testGetFoodStoresByStationIds
129	testGetFoodStoresByStationIds1	129	testGetFoodStoresByStationIds1
130	testGetFoodStoresByStationIds2	130	testGetFoodStoresByStationIds2
131	testGetFoodStoresOfStation	131	testGetFoodStoresOfStation
132	testGetLeftTicketOfInterva2	132	testGetLeftTicketOfInterva2
133	testGetLeftTicketOfInterval	133	testGetLeftTicketOfInterval
134	testGetMinStation	134	testGetMinStation
135	testGetMinStopStations	135	testGetMinStopStations
136	testGetOrderById	136	testGetOrderById
137	testGetOrderById1	137	testGetOrderById1
138	testGetOrderById2	138	testGetOrderById2
139	testGetOrderPrice	139	testGetOrderPrice
140	testGetOrderPrice1	140	testGetOrderPrice1
141	testGetOrderPrice2	141	testGetOrderPrice2
142	testGetPriceByWeightAndRegion	142	testGetPriceByWeightAndRegion
143	testGetPriceConfig	143	testGetPriceConfig
144	testGetPriceInfo	144	testGetPriceInfo
145	testGetQuickest	145	testGetQuickest
146	testGetQuickestRoutes	146	testGetQuickestRoutes
147	testGetRouteById1	147	testGetRouteById1
148	testGetRouteById2	148	testGetRouteById2
149	testGetRouteByTripId	149	testGetRouteByTripId
150	testGetRouteByTripId1	150	testGetRouteByTripId1
151	testGetRouteByTripId2	151	testGetRouteByTripId2
152	testGetSoldTickets1	152	testGetSoldTickets1
153	testGetSoldTickets2	153	testGetSoldTickets2
154	testGetTicketListByDateAndTripId	154	testGetTicketListByDateAndTripId
155	testGetToken1	155	testGetToken1
156	testGetToken2	156	testGetToken2
157	testGetTrainFoodOfTrip	157	testGetTrainFoodOfTrip
158	testGetTrainTypeByTripId	158	testGetTrainTypeByTripId
159	testGetTransferResult	159	testGetTransferResult
160	testGetTransferSearch	160	testGetTransferSearch
161	testGetTripAllDetailInfo	161	testGetTripAllDetailInfo
162	testGetTripByRoute2	162	testGetTripByRoute2
163	testGetTripsByRouteId	163	testGetTripsByRouteId
164	testHome	164	testHome
165	testInitOrder1	165	testInitOrder1
166	testInitOrder2	166	testInitOrder2
167	testInitPayment1	167	testInitPayment1
168	testInitPayment2	168	testInitPayment2
169	testInsertConsign	169	testInsertConsign
170	testListFoodStores1	170	testListFoodStores1
171	testListFoodStoresByStationId1	171	testListFoodStoresByStationId1
172	testListFoodStoresByStationId2	172	testListFoodStoresByStationId2
173	testListTrainFood1	173	testListTrainFood1
174	testListTrainFoodByTripId1	174	testListTrainFoodByTripId1
175	testListTrainFoodByTripId2	175	testListTrainFoodByTripId2
176	testModifyAssurance	176	testModifyAssurance
177	testModifyConfig	177	testModifyConfig
178	testModifyContact	178	testModifyContact
179	testModifyContacts	179	testModifyContacts
180	testModifyOrder	180	testModifyOrder
181	testModifyOrder1	181	testModifyOrder1
182	testModifyOrder2	182	testModifyOrder2
183	testModifyPrice	183	testModifyPrice
184	testModifyPriceConfig	184	testModifyPriceConfig
185	testModifySecurityConfig1	185	testModifySecurityConfig1
186	testModifySecurityConfig2	186	testModifySecurityConfig2
187	testModifyStation	187	testModifyStation
188	testModifyTrain	188	testModifyTrain
189	testOrderCancelSuccess	189	testOrderCancelSuccess
190	testOrderChangedSuccess	190	testOrderChangedSuccess
191	testOrderCreateSuccess	191	testOrderCreateSuccess
192	testPay	192	testPay
193	testPay1	193	testPay1

	TP2		**TP1**
TP Number	**TP Name**	**TP Number**	**TP Name**
194	testPay2	194	testPay2
195	testPayDifference	195	testPayDifference
196	testPayOrder	196	testPayOrder
197	testPayOrder1	197	testPayOrder1
198	testPayOrder2	198	testPayOrder2
199	testPreserve	199	testPreserve
200	testPreserveSuccess	200	testPreserveSuccess
201	testQuery	201	testQuery
202	testQuery1	202	testQuery1
203	testQuery2	203	testQuery2
204	testQueryAccount	204	testQueryAccount
205	testQueryAddMoney	205	testQueryAddMoney
206	testQueryAddMoney1	206	testQueryAddMoney1
207	testQueryAll	207	testQueryAll
208	testQueryAll1	208	testQueryAll1
209	testQueryAlreadySoldOrders	209	testQueryAlreadySoldOrders
210	testQueryByConsignee1	210	testQueryByConsignee1
211	testQueryByConsignee2	211	testQueryByConsignee2
212	testQueryById	212	testQueryById
213	testQueryByStartAndTerminal	213	testQueryByStartAndTerminal
214	testQueryForStationId	214	testQueryForStationId
215	testQueryForTravel	215	testQueryForTravel
216	testQueryInfo1	216	testQueryInfo1
217	testQueryInfo2	217	testQueryInfo2
218	testQueryOrders	218	testQueryOrders
219	testQueryOrdersForRefresh	219	testQueryOrdersForRefresh
220	testQueryPayment	220	testQueryPayment
221	testQueryPayment1	221	testQueryPayment1
222	testQueryTrainType	222	testQueryTrainType
223	testRebook	223	testRebook
224	testRetrieve	224	testRetrieve
225	testRetrieve1	225	testRetrieve1
226	testRetrieve2	226	testRetrieve2
227	testSaveChanges1	227	testSaveChanges1
228	testSaveChanges2	228	testSaveChanges2
229	testSaveOrderInfo	229	testSaveOrderInfo
230	testSaveUser	230	testSaveUser
231	testSearchMinStopStations	231	testSearchMinStopStations
232	testSearchQuickestResult	232	testSearchQuickestResult
233	testSendEmail	233	testSendEmail
234	testTicketCollect1	234	testTicketCollect1
235	testTicketCollect2	235	testTicketCollect2
236	testTicketExecute1	236	testTicketExecute1
237	testTicketExecute2	237	testTicketExecute2
238	testUpdate	238	testUpdate
239	testUpdate1	239	testUpdate1
240	testUpdate2	240	testUpdate2
241	testUpdateConfig	241	testUpdateConfig
242	testUpdateConsign	242	testUpdateConsign
243	testUpdateFoodOrder	243	testUpdateFoodOrder
244	testUpdateFoodOrder1	244	testUpdateFoodOrder1
245	testUpdateOrder	245	testUpdateOrder
246	testUpdateOrder1	246	testUpdateOrder1
247	testUpdateOrder2	247	testUpdateOrder2
248	testUpdatePriceConfig1	248	testUpdatePriceConfig1
249	testUpdatePriceConfig2	249	testUpdatePriceConfig2
250	testUpdateTravel	250	testUpdateTravel
251	testUpdateTravel1	251	testUpdateTravel1
252	testUpdateTravel2	252	testUpdateTravel2
253	testUpdateTrip	253	testUpdateTrip
254	testUpdateUser	254	testUpdateUser

Table A.7: *Domain*: `invokes`; *Criterion*: `nonemptySubSeq`. Refers to Figure A.45.

	TP2		**TP1**
TP Number	**TP Name**	**TP Number**	**TP Name**
1	testAddConfig	1	testAddConfig
2	testAddContact	2	testAddContact
3	testAddContacts	3	testAddContacts
4	testAddCreateNewOrder	4	testAddCreateNewOrder
5	testAddMoney	5	testAddMoney
6	testAddMoney1	6	testAddMoney1
7	testAddMoney2	7	testAddMoney2
8	testAddNewOrder1	8	testAddNewOrder1
9	testAddNewOrder2	9	testAddNewOrder2
10	testAddNewSecurityConfig1	10	testAddNewSecurityConfig1
11	testAddNewSecurityConfig2	11	testAddNewSecurityConfig2
12	testAddOrder	12	testAddOrder
13	testAddOrder1	13	testAddOrder1
14	testAddOrder2	14	testAddOrder2
15	testAddPrice	15	testAddPrice
16	testAddRoute	16	testAddStation
17	testAddStation	17	testAddTrain
18	testAddTrain	18	testAddTravel
19	testAddTravel	19	testAddTravel1
20	testAddTravel1	20	testAddTravel2
21	testAddTravel2	21	testAddTravel3
22	testAddTravel3	22	testAddTravel4
23	testAddTravel4	23	testAddUser
24	testAddUser	24	testAdminQueryAll
25	testAdminQueryAll	25	testAdminQueryAll1
26	testAdminQueryAll1	26	testCancelOrder1
27	testAlterOrder1	27	testCancelOrder2
28	testCancelOrder1	28	testCheck
29	testCancelOrder2	29	testCheckSecurityAboutOrder
30	testCheck	30	testCheckStationExists
31	testCheckSecurityAboutOrder	31	testCollectTicket
32	testCheckStationExists	32	testCreate
33	testCollectTicket	33	testCreate1
34	testCreate	34	testCreate2
35	testCreate1	35	testCreateConfig
36	testCreate2	36	testCreateContacts1
37	testCreateAccount	37	testCreateContacts2
38	testCreateAccount1	38	testCreateFoodOrder
39	testCreateAccount2	39	testCreateFoodOrder1

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
40	testCreateAndModify2	40	testCreateFoodStore1
41	testCreateAndModify3	41	testCreateFoodStore2
42	testCreateConfig	42	testCreateNewContacts
43	testCreateContacts1	43	testCreateNewContactsAdmin
44	testCreateContacts2	44	testCreateNewOrder
45	testCreateFoodOrder	45	testCreateNewPriceConfig1
46	testCreateFoodOrder1	46	testCreateTrainFood1
47	testCreateFoodStore1	47	testCreateTrainFood2
48	testCreateFoodStore2	48	testCreateTrip
49	testCreateNewContacts	49	testDelete
50	testCreateNewContactsAdmin	50	testDelete1
51	testCreateNewOrder	51	testDelete2
52	testCreateNewPriceConfig1	52	testDeleteConfig
53	testCreateTrainFood1	53	testDeleteContact
54	testCreateTrainFood2	54	testDeleteContacts
55	testCreateTrip	55	testDeleteFoodOrder
56	testDelete	56	testDeleteOrder
57	testDelete1	57	testDeleteOrder1
58	testDelete2	58	testDeleteOrder2
59	testDeleteConfig	59	testDeletePrice
60	testDeleteContact	60	testDeletePriceConfig1
61	testDeleteContacts	61	testDeletePriceConfig2
62	testDeleteFoodOrder	62	testDeleteRoute
63	testDeleteOrder	63	testDeleteRoute1
64	testDeleteOrder1	64	testDeleteStation
65	testDeleteOrder2	65	testDeleteTrain
66	testDeletePrice	66	testDeleteTravel
67	testDeletePriceConfig1	67	testDeleteTravel1
68	testDeletePriceConfig2	68	testDeleteTravel2
69	testDeleteRoute	69	testDeleteTrip
70	testDeleteRoute2	70	testDeleteUser
71	testDeleteStation	71	testDistributeSeat1
72	testDeleteTrain	72	testDistributeSeat2
73	testDeleteTravel	73	testDrawBack
74	testDeleteTravel1	74	testDrawBack1
75	testDeleteTravel2	75	testDrawBack2
76	testDeleteTrip	76	testExecuteTicket
77	testDeleteUser	77	testFindAllFoodOrder
78	testDistributeSeat1	78	testFindAllFoodOrder1
79	testDistributeSeat2	79	testFindAllOrder
80	testDrawBack	80	testFindAllPriceConfig2
81	testDrawBack1	81	testFindAllSecurityConfig
82	testDrawBack2	82	testFindAllSecurityConfig1
83	testExecuteTicket	83	testFindByConsignee
84	testFindAllFoodOrder	84	testFindById
85	testFindAllOrder	85	testFindByRouteIdAndTrainType1
86	testFindAllSecurityConfig	86	testFindFoodOrderByOrderId
87	testFindByConsignee	87	testGetAllAssuranceType
88	testFindById	88	testGetAllAssurances
89	testFindByRouteIdAndTrainType1	89	testGetAllConfigs
90	testFindByRouteIdAndTrainType2	90	testGetAllContacts
91	testFindFoodOrderByOrderId	91	testGetAllContacts1
92	testGetAllAssuranceType	92	testGetAllFood
93	testGetAllAssurances	93	testGetAllFoodStores
94	testGetAllConfigs	94	testGetAllOrders
95	testGetAllContacts	95	testGetAllOrders2
96	testGetAllFood	96	testGetAllPrices
97	testGetAllFoodStores	97	testGetAllRoutes
98	testGetAllOrders	98	testGetAllRoutes1
99	testGetAllOrders2	99	testGetAllStations
100	testGetAllPrices	100	testGetAllTrains
101	testGetAllRoutes	101	testGetAllTravels
102	testGetAllStations	102	testGetAllUser
103	testGetAllTrains	103	testGetAllUsers
104	testGetAllTravels	104	testGetByCheapest
105	testGetAllUser	105	testGetByMinStation
106	testGetAllUsers	106	testGetByQuickest
107	testGetByCheapest	107	testGetCheapestRoutes
108	testGetByMinStation	108	testGetFoodStoresByStationIds
109	testGetByQuickest	109	testGetFoodStoresByStationIds2
110	testGetCheapest	110	testGetFoodStoresOfStation
111	testGetCheapestRoutes	111	testGetLeftTicketOfInterval
112	testGetFoodStoresByStationIds	112	testGetMinStopStations
113	testGetFoodStoresByStationIds1	113	testGetOrderById
114	testGetFoodStoresByStationIds2	114	testGetOrderById1
115	testGetFoodStoresOfStation	115	testGetOrderById2
116	testGetLeftTicketOfInterva2	116	testGetOrderPrice
117	testGetLeftTicketOfInterval	117	testGetOrderPrice1
118	testGetMinStation	118	testGetOrderPrice2
119	testGetMinStopStations	119	testGetPriceConfig
120	testGetOrderById	120	testGetPriceInfo
121	testGetOrderById1	121	testGetQuickestRoutes
122	testGetOrderById2	122	testGetRouteById1
123	testGetOrderPrice	123	testGetRouteByTripId
124	testGetOrderPrice1	124	testGetRouteByTripId1
125	testGetOrderPrice2	125	testGetRouteByTripId2
126	testGetPriceConfig	126	testGetSoldTickets2
127	testGetPriceInfo	127	testGetTicketListByDateAndTripId
128	testGetQuickest	128	testGetTrainFoodOfTrip
129	testGetQuickestRoutes	129	testGetTrainTypeByTripId
130	testGetRouteById1	130	testGetTripAllDetailInfo
131	testGetRouteById2	131	testGetTripByRoute2
132	testGetRouteByTripId	132	testGetTripsByRouteId
133	testGetRouteByTripId1	133	testHome
134	testGetRouteByTripId2	134	testInitOrder1
135	testGetSoldTickets1	135	testInitOrder2
136	testGetSoldTickets2	136	testInitPayment1
137	testGetTicketListByDateAndTripId	137	testInitPayment2
138	testGetToken1	138	testInsertConsign
139	testGetToken2	139	testListFoodStores1
140	testGetTrainFoodOfTrip	140	testListFoodStoresByStationId2
141	testGetTrainTypeByTripId	141	testListTrainFood1
142	testGetTransferResult	142	testListTrainFoodByTripId2
143	testGetTransferSearch	143	testModifyAssurance
144	testGetTripAllDetailInfo	144	testModifyConfig
145	testGetTripByRoute2	145	testModifyContact
146	testGetTripsByRouteId	146	testModifyContacts
147	testHome	147	testModifyOrder
148	testInitOrder1	148	testModifyOrder1
149	testInitOrder2	149	testModifyOrder2

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
150	testInitPayment1	150	testModifyPrice
151	testInitPayment2	151	testModifySecurityConfig1
152	testInsertConsign	152	testModifySecurityConfig2
153	testListFoodStoresByStationId1	153	testModifyStation
154	testListFoodStoresByStationId2	154	testModifyTrain
155	testListTrainFoodByTripId1	155	testOrderCancelSuccess
156	testListTrainFoodByTripId2	156	testOrderChangedSuccess
157	testModifyAssurance	157	testOrderCreateSuccess
158	testModifyConfig	158	testPay
159	testModifyContact	159	testPay1
160	testModifyContacts	160	testPay2
161	testModifyOrder	161	testPayDifference
162	testModifyOrder1	162	testPayOrder
163	testModifyOrder2	163	testPayOrder1
164	testModifyPrice	164	testPayOrder2
165	testModifyPriceConfig	165	testPreserveSuccess
166	testModifySecurityConfig1	166	testQuery
167	testModifySecurityConfig2	167	testQuery1
168	testModifyStation	168	testQuery2
169	testModifyTrain	169	testQueryAccount
170	testOrderCancelSuccess	170	testQueryAddMoney
171	testOrderChangedSuccess	171	testQueryAddMoney1
172	testOrderCreateSuccess	172	testQueryAll
173	testPay	173	testQueryAll1
174	testPay1	174	testQueryAlreadySoldOrders
175	testPay2	175	testQueryByConsignee2
176	testPayDifference	176	testQueryById
177	testPayOrder	177	testQueryByStartAndTerminal
178	testPayOrder1	178	testQueryForStationId
179	testPayOrder2	179	testQueryInfo1
180	testPreserve	180	testQueryOrders
181	testPreserveSuccess	181	testQueryOrdersForRefresh
182	testQuery	182	testQueryPayment
183	testQuery1	183	testQueryPayment1
184	testQuery2	184	testQueryTrainType
185	testQueryAccount	185	testRebook
186	testQueryAddMoney	186	testRetrieve
187	testQueryAll	187	testRetrieve1
188	testQueryAll1	188	testRetrieve2
189	testQueryAlreadySoldOrders	189	testSaveChanges1
190	testQueryByConsignee1	190	testSaveChanges2
191	testQueryByConsignee2	191	testSaveOrderInfo
192	testQueryById	192	testSearchQuickestResult
193	testQueryByStartAndTerminal	193	testUpdate
194	testQueryForStationId	194	testUpdate1
195	testQueryForTravel	195	testUpdate2
196	testQueryInfo2	196	testUpdateConfig
197	testQueryOrders	197	testUpdateConsign
198	testQueryOrdersForRefresh	198	testUpdateFoodOrder
199	testQueryPayment	199	testUpdateFoodOrder1
200	testQueryPayment1	200	testUpdateOrder
201	testQueryTrainType	201	testUpdateOrder1
202	testRebook	202	testUpdateOrder2
203	testRetrieve	203	testUpdatePriceConfig1
204	testRetrieve1	204	testUpdatePriceConfig2
205	testRetrieve2	205	testUpdateTravel
206	testSaveChanges1	206	testUpdateTravel1
207	testSaveChanges2	207	testUpdateTravel2
208	testSaveOrderInfo	208	testUpdateTrip
209	testSaveUser	209	testUpdateUser
210	testSearchMinStopStations		
211	testSearchQuickestResult		
212	testUpdate		
213	testUpdate1		
214	testUpdate2		
215	testUpdateConfig		
216	testUpdateConsign		
217	testUpdateFoodOrder		
218	testUpdateFoodOrder1		
219	testUpdateOrder		
220	testUpdateOrder1		
221	testUpdateOrder2		
222	testUpdatePriceConfig1		
223	testUpdatePriceConfig2		
224	testUpdateTravel		
225	testUpdateTravel1		
226	testUpdateTravel2		
227	testUpdateTrip		
228	testUpdateUser		

Table A.8: *Domain*: `invokes`; *Criterion*: `nonemptySubSet`. Refers to Figure A.46.

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
1	testAddConfig	1	testAddConfig
2	testAddContact	2	testAddContact
3	testAddContacts	3	testAddContacts
4	testAddCreateNewOrder	4	testAddCreateNewOrder
5	testAddMoney	5	testAddMoney
6	testAddMoney1	6	testAddMoney1
7	testAddMoney2	7	testAddMoney2
8	testAddNewOrder1	8	testAddNewOrder1
9	testAddNewOrder2	9	testAddNewOrder2
10	testAddNewSecurityConfig1	10	testAddNewSecurityConfig1
11	testAddNewSecurityConfig2	11	testAddNewSecurityConfig2
12	testAddOrder	12	testAddOrder
13	testAddOrder1	13	testAddOrder1
14	testAddOrder2	14	testAddOrder2
15	testAddPrice	15	testAddPrice
16	testAddRoute	16	testAddStation
17	testAddStation	17	testAddTrain
18	testAddTrain	18	testAddTravel
19	testAddTravel	19	testAddTravel1
20	testAddTravel1	20	testAddTravel2
21	testAddTravel2	21	testAddTravel3

	TP2		TP1
TP Number	TP Name	TP Number	TP Name
22	testAddTravel3	22	testAddTravel4
23	testAddTravel4	23	testAddUser
24	testAddUser	24	testAdminQueryAll
25	testAdminQueryAll	25	testAdminQueryAll1
26	testAdminQueryAll1	26	testCancelOrder1
27	testAlterOrder1	27	testCancelOrder2
28	testCancelOrder1	28	testCheck
29	testCancelOrder2	29	testCheckSecurityAboutOrder
30	testCheck	30	testCheckStationExists
31	testCheckSecurityAboutOrder	31	testCollectTicket
32	testCheckStationExists	32	testCreate
33	testCollectTicket	33	testCreate1
34	testCreate	34	testCreate2
35	testCreate1	35	testCreateConfig
36	testCreate2	36	testCreateContacts1
37	testCreateAccount	37	testCreateContacts2
38	testCreateAccount1	38	testCreateFoodOrder
39	testCreateAccount2	39	testCreateFoodOrder1
40	testCreateAndModify2	40	testCreateFoodStore1
41	testCreateAndModify3	41	testCreateFoodStore2
42	testCreateConfig	42	testCreateNewContacts
43	testCreateContacts1	43	testCreateNewContactsAdmin
44	testCreateContacts2	44	testCreateNewOrder
45	testCreateFoodOrder	45	testCreateNewPriceConfig1
46	testCreateFoodOrder1	46	testCreateTrainFood1
47	testCreateFoodStore1	47	testCreateTrainFood2
48	testCreateFoodStore2	48	testCreateTrip
49	testCreateNewContacts	49	testDelete
50	testCreateNewContactsAdmin	50	testDelete1
51	testCreateNewOrder	51	testDelete2
52	testCreateNewPriceConfig1	52	testDeleteConfig
53	testCreateTrainFood1	53	testDeleteContact
54	testCreateTrainFood2	54	testDeleteContacts
55	testCreateTrip	55	testDeleteFoodOrder
56	testDelete	56	testDeleteOrder
57	testDelete1	57	testDeleteOrder1
58	testDelete2	58	testDeleteOrder2
59	testDeleteConfig	59	testDeletePrice
60	testDeleteContact	60	testDeletePriceConfig1
61	testDeleteContacts	61	testDeletePriceConfig2
62	testDeleteFoodOrder	62	testDeleteRoute
63	testDeleteOrder	63	testDeleteRoute1
64	testDeleteOrder1	64	testDeleteStation
65	testDeleteOrder2	65	testDeleteTrain
66	testDeletePrice	66	testDeleteTravel
67	testDeletePriceConfig1	67	testDeleteTravel1
68	testDeletePriceConfig2	68	testDeleteTravel2
69	testDeleteRoute	69	testDeleteTrip
70	testDeleteRoute2	70	testDeleteUser
71	testDeleteStation	71	testDistributeSeat1
72	testDeleteTrain	72	testDistributeSeat2
73	testDeleteTravel	73	testDrawBack
74	testDeleteTravel1	74	testDrawBack1
75	testDeleteTravel2	75	testDrawBack2
76	testDeleteTrip	76	testExecuteTicket
77	testDeleteUser	77	testFindAllFoodOrder
78	testDistributeSeat1	78	testFindAllFoodOrder1
79	testDistributeSeat2	79	testFindAllOrder
80	testDrawBack	80	testFindAllPriceConfig2
81	testDrawBack1	81	testFindAllSecurityConfig
82	testDrawBack2	82	testFindAllSecurityConfig1
83	testExecuteTicket	83	testFindByConsignee
84	testFindAllFoodOrder	84	testFindById
85	testFindAllOrder	85	testFindByRouteIdAndTrainType1
86	testFindAllSecurityConfig	86	testFindFoodOrderByOrderId
87	testFindByConsignee	87	testGetAllAssuranceType
88	testFindById	88	testGetAllAssurances
89	testFindByRouteIdAndTrainType1	89	testGetAllConfigs
90	testFindByRouteIdAndTrainType2	90	testGetAllContacts
91	testFindFoodOrderByOrderId	91	testGetAllContacts1
92	testGetAllAssuranceType	92	testGetAllFood
93	testGetAllAssurances	93	testGetAllFoodStores
94	testGetAllConfigs	94	testGetAllOrders
95	testGetAllContacts	95	testGetAllOrders2
96	testGetAllFood	96	testGetAllPrices
97	testGetAllFoodStores	97	testGetAllRoutes
98	testGetAllOrders	98	testGetAllRoutes1
99	testGetAllOrders2	99	testGetAllStations
100	testGetAllPrices	100	testGetAllTrains
101	testGetAllRoutes	101	testGetAllTravels
102	testGetAllStations	102	testGetAllUser
103	testGetAllTrains	103	testGetAllUsers
104	testGetAllTravels	104	testGetByCheapest
105	testGetAllUser	105	testGetByMinStation
106	testGetAllUsers	106	testGetByQuickest
107	testGetByCheapest	107	testGetCheapestRoutes
108	testGetByMinStation	108	testGetFoodStoresByStationIds
109	testGetByQuickest	109	testGetFoodStoresByStationIds2
110	testGetCheapest	110	testGetFoodStoresOfStation
111	testGetCheapestRoutes	111	testGetLeftTicketOfInterval
112	testGetFoodStoresByStationIds	112	testGetMinStopStations
113	testGetFoodStoresByStationIds1	113	testGetOrderById
114	testGetFoodStoresByStationIds2	114	testGetOrderById1
115	testGetFoodStoresOfStation	115	testGetOrderById2
116	testGetLeftTicketOfInterva2	116	testGetOrderPrice
117	testGetLeftTicketOfInterval	117	testGetOrderPrice1
118	testGetMinStation	118	testGetOrderPrice2
119	testGetMinStopStations	119	testGetPriceConfig
120	testGetOrderById	120	testGetPriceInfo
121	testGetOrderById1	121	testGetQuickestRoutes
122	testGetOrderById2	122	testGetRouteById1
123	testGetOrderPrice	123	testGetRouteByTripId
124	testGetOrderPrice1	124	testGetRouteByTripId1
125	testGetOrderPrice2	125	testGetRouteByTripId2
126	testGetPriceConfig	126	testGetSoldTickets2
127	testGetPriceInfo	127	testGetTicketListByDateAndTripId
128	testGetQuickest	128	testGetTrainFoodOfTrip
129	testGetQuickestRoutes	129	testGetTrainTypeByTripId
130	testGetRouteById1	130	testGetTripAllDetailInfo
131	testGetRouteById2	131	testGetTripByRoute2

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
132	testGetRouteByTripId	132	testGetTripsByRouteId
133	testGetRouteByTripId1	133	testHome
134	testGetRouteByTripId2	134	testInitOrder1
135	testGetSoldTickets1	135	testInitOrder2
136	testGetSoldTickets2	136	testInitPayment1
137	testGetTicketListByDateAndTripId	137	testInitPayment2
138	testGetToken1	138	testInsertConsign
139	testGetToken2	139	testListFoodStores1
140	testGetTrainFoodOfTrip	140	testListFoodStoresByStationId2
141	testGetTrainTypeByTripId	141	testListTrainFood1
142	testGetTransferResult	142	testListTrainFoodByTripId2
143	testGetTransferSearch	143	testModifyAssurance
144	testGetTripAllDetailInfo	144	testModifyConfig
145	testGetTripByRoute2	145	testModifyContact
146	testGetTripsByRouteId	146	testModifyContacts
147	testHome	147	testModifyOrder
148	testInitOrder1	148	testModifyOrder1
149	testInitOrder2	149	testModifyOrder2
150	testInitPayment1	150	testModifyPrice
151	testInitPayment2	151	testModifySecurityConfig1
152	testInsertConsign	152	testModifySecurityConfig2
153	testListFoodStoresByStationId1	153	testModifyStation
154	testListFoodStoresByStationId2	154	testModifyTrain
155	testListTrainFoodByTripId1	155	testOrderCancelSuccess
156	testListTrainFoodByTripId2	156	testOrderChangedSuccess
157	testModifyAssurance	157	testOrderCreateSuccess
158	testModifyConfig	158	testPay
159	testModifyContact	159	testPay1
160	testModifyContacts	160	testPay2
161	testModifyOrder	161	testPayDifference
162	testModifyOrder1	162	testPayOrder
163	testModifyOrder2	163	testPayOrder1
164	testModifyPrice	164	testPayOrder2
165	testModifyPriceConfig	165	testPreserveSuccess
166	testModifySecurityConfig1	166	testQuery
167	testModifySecurityConfig2	167	testQuery1
168	testModifyStation	168	testQuery2
169	testModifyTrain	169	testQueryAccount
170	testOrderCancelSuccess	170	testQueryAddMoney
171	testOrderChangedSuccess	171	testQueryAddMoney1
172	testOrderCreateSuccess	172	testQueryAll
173	testPay	173	testQueryAll1
174	testPay1	174	testQueryAlreadySoldOrders
175	testPay2	175	testQueryByConsignee2
176	testPayDifference	176	testQueryById
177	testPayOrder	177	testQueryByStartAndTerminal
178	testPayOrder1	178	testQueryForStationId
179	testPayOrder2	179	testQueryInfo1
180	testPreserve	180	testQueryOrders
181	testPreserveSuccess	181	testQueryOrdersForRefresh
182	testQuery	182	testQueryPayment
183	testQuery1	183	testQueryPayment1
184	testQuery2	184	testQueryTrainType
185	testQueryAccount	185	testRebook
186	testQueryAddMoney	186	testRetrieve
187	testQueryAll	187	testRetrieve1
188	testQueryAll1	188	testRetrieve2
189	testQueryAlreadySoldOrders	189	testSaveChanges1
190	testQueryByConsignee1	190	testSaveChanges2
191	testQueryByConsignee2	191	testSaveOrderInfo
192	testQueryById	192	testSearchMinStopStations
193	testQueryByStartAndTerminal	193	testSearchQuickestResult
194	testQueryForStationId	194	testUpdate
195	testQueryForTravel	195	testUpdate1
196	testQueryInfo2	196	testUpdate2
197	testQueryOrders	197	testUpdateConfig
198	testQueryOrdersForRefresh	198	testUpdateConsign
199	testQueryPayment	199	testUpdateFoodOrder
200	testQueryPayment1	200	testUpdateFoodOrder1
201	testQueryTrainType	201	testUpdateOrder
202	testRebook	202	testUpdateOrder1
203	testRetrieve	203	testUpdateOrder2
204	testRetrieve1	204	testUpdatePriceConfig1
205	testRetrieve2	205	testUpdatePriceConfig2
206	testSaveChanges1	206	testUpdateTravel
207	testSaveChanges2	207	testUpdateTravel1
208	testSaveOrderInfo	208	testUpdateTravel2
209	testSaveUser	209	testUpdateTrip
210	testSearchMinStopStations	210	testUpdateUser
211	testSearchQuickestResult		
212	testUpdate		
213	testUpdate1		
214	testUpdate2		
215	testUpdateConfig		
216	testUpdateConsign		
217	testUpdateFoodOrder		
218	testUpdateFoodOrder1		
219	testUpdateOrder		
220	testUpdateOrder1		
221	testUpdateOrder2		
222	testUpdatePriceConfig1		
223	testUpdatePriceConfig2		
224	testUpdateTravel		
225	testUpdateTravel1		
226	testUpdateTravel2		
227	testUpdateTrip		
228	testUpdateUser		

Appendix A.5.2. Concrete Execution

Table A.9: *Domain*: `invokes`; *Criterion*: `nonemptyCommonSeq`. Refers to Figure A.53.

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
1	tesCreate2	1	tesCreate2

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
2	testAddConfig	2	testAddConfig
3	testAddContact	3	testAddContact
4	testAddContacts	4	testAddContacts
5	testAddCreateNewOrder	5	testAddCreateNewOrder
6	testAddMoney	6	testAddMoney
7	testAddMoney1	7	testAddMoney1
8	testAddMoney2	8	testAddMoney2
9	testAddNewOrder1	9	testAddNewOrder1
10	testAddNewOrder2	10	testAddNewOrder2
11	testAddNewSecurityConfig1	11	testAddNewSecurityConfig1
12	testAddNewSecurityConfig2	12	testAddNewSecurityConfig2
13	testAddOrder	13	testAddOrder
14	testAddOrder1	14	testAddOrder1
15	testAddOrder2	15	testAddOrder2
16	testAddPrice	16	testAddPrice
17	testAddRoute	17	testAddRoute
18	testAddStation	18	testAddStation
19	testAddTrain	19	testAddTrain
20	testAddTravel	20	testAddTravel
21	testAddTravel1	21	testAddTravel1
22	testAddTravel2	22	testAddTravel2
23	testAddTravel3	23	testAddTravel3
24	testAddTravel4	24	testAddTravel4
25	testAddUser	25	testAddUser
26	testAdminQueryAll	26	testAdminQueryAll
27	testAdminQueryAll1	27	testAdminQueryAll1
28	testAdminQueryAll2	28	testAdminQueryAll2
29	testAlterOrder1	29	testAlterOrder1
30	testAlterOrder2	30	testAlterOrder2
31	testCalculate	31	testCalculate
32	testCalculateRefund1	32	testCalculateRefund1
33	testCalculateRefund2	33	testCalculateRefund2
34	testCancelOrder1	34	testCancelOrder1
35	testCancelOrder2	35	testCancelOrder2
36	testCancelTicket	36	testCancelTicket
37	testCheck	37	testCheck
38	testCheckSecurityAboutOrder	38	testCheckSecurityAboutOrder
39	testCheckStationExists	39	testCheckStationExists
40	testCollectTicket	40	testCollectTicket
41	testCreate	41	testCreate
42	testCreate1	42	testCreate1
43	testCreate2	43	testCreate2
44	testCreate3	44	testCreate3
45	testCreateAccount	45	testCreateAccount
46	testCreateAccount1	46	testCreateAccount1
47	testCreateAccount2	47	testCreateAccount2
48	testCreateAndModify1	48	testCreateAndModify1
49	testCreateAndModify2	49	testCreateAndModify2
50	testCreateAndModify3	50	testCreateAndModify3
51	testCreateAndModifyPrice1	51	testCreateAndModifyPrice1
52	testCreateAndModifyPrice2	52	testCreateAndModifyPrice2
53	testCreateAndModifyRoute	53	testCreateAndModifyRoute
54	testCreateConfig	54	testCreateConfig
55	testCreateContacts1	55	testCreateContacts1
56	testCreateContacts2	56	testCreateContacts2
57	testCreateDefaultAuthUser	57	testCreateDefaultAuthUser
58	testCreateDefaultUser	58	testCreateDefaultUser
59	testCreateFoodOrder	59	testCreateFoodOrder
60	testCreateFoodOrder1	60	testCreateFoodOrder1
61	testCreateFoodOrder2	61	testCreateFoodOrder2
62	testCreateFoodStore1	62	testCreateFoodStore1
63	testCreateFoodStore2	63	testCreateFoodStore2
64	testCreateNewAssurance	64	testCreateNewAssurance
65	testCreateNewContacts	65	testCreateNewContacts
66	testCreateNewContactsAdmin	66	testCreateNewContactsAdmin
67	testCreateNewOrder	67	testCreateNewOrder
68	testCreateNewPriceConfig1	68	testCreateNewPriceConfig1
69	testCreateNewPriceConfig2	69	testCreateNewPriceConfig2
70	testCreateTrainFood1	70	testCreateTrainFood1
71	testCreateTrainFood2	71	testCreateTrainFood2
72	testCreateTrip	72	testCreateTrip
73	testDelete	73	testDelete
74	testDelete1	74	testDelete1
75	testDelete2	75	testDelete2
76	testDeleteAssurance	76	testDeleteAssurance
77	testDeleteAssuranceByOrderId	77	testDeleteAssuranceByOrderId
78	testDeleteById1	78	testDeleteById1
79	testDeleteById2	79	testDeleteById2
80	testDeleteByOrderId1	80	testDeleteByOrderId1
81	testDeleteByOrderId2	81	testDeleteByOrderId2
82	testDeleteByUserId	82	testDeleteByUserId
83	testDeleteConfig	83	testDeleteConfig
84	testDeleteContact	84	testDeleteContact
85	testDeleteContacts	85	testDeleteContacts
86	testDeleteFoodOrder	86	testDeleteFoodOrder
87	testDeleteFoodOrder1	87	testDeleteFoodOrder1
88	testDeleteFoodOrder2	88	testDeleteFoodOrder2
89	testDeleteOrder	89	testDeleteOrder
90	testDeleteOrder1	90	testDeleteOrder1
91	testDeleteOrder2	91	testDeleteOrder2
92	testDeletePrice	92	testDeletePrice
93	testDeletePriceConfig1	93	testDeletePriceConfig1
94	testDeletePriceConfig2	94	testDeletePriceConfig2
95	testDeleteRoute	95	testDeleteRoute
96	testDeleteRoute1	96	testDeleteRoute1
97	testDeleteRoute2	97	testDeleteRoute2
98	testDeleteSecurityConfig1	98	testDeleteSecurityConfig1
99	testDeleteSecurityConfig2	99	testDeleteSecurityConfig2
100	testDeleteStation	100	testDeleteStation
101	testDeleteTrain	101	testDeleteTrain
102	testDeleteTravel	102	testDeleteTravel
103	testDeleteTravel1	103	testDeleteTravel1
104	testDeleteTravel2	104	testDeleteTravel2
105	testDeleteTrip	105	testDeleteTrip
106	testDeleteUser	106	testDeleteUser
107	testDeleteUser1	107	testDeleteUser1
108	testDeleteUser2	108	testDeleteUser2
109	testDeleteUserAuth	109	testDeleteUserAuth
110	testDeleteUserById	110	testDeleteUserById
111	testDipatchSeat	111	testDipatchSeat

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
112	testDistributeSeat1	112	testDistributeSeat1
113	testDistributeSeat2	113	testDistributeSeat2
114	testDrawBack	114	testDrawBack
115	testDrawBack1	115	testDrawBack1
116	testDrawBack2	116	testDrawBack2
117	testDrawbackMoney	117	testDrawbackMoney
118	testExecuteTicket	118	testExecuteTicket
119	testExist1	119	testExist1
120	testExist2	120	testExist2
121	testFindAllFoodOrder	121	testFindAllFoodOrder
122	testFindAllFoodOrder1	122	testFindAllFoodOrder1
123	testFindAllFoodOrder2	123	testFindAllFoodOrder2
124	testFindAllOrder	124	testFindAllOrder
125	testFindAllPriceConfig1	125	testFindAllPriceConfig1
126	testFindAllPriceConfig2	126	testFindAllPriceConfig2
127	testFindAllSecurityConfig	127	testFindAllSecurityConfig
128	testFindAllSecurityConfig1	128	testFindAllSecurityConfig1
129	testFindAllSecurityConfig2	129	testFindAllSecurityConfig2
130	testFindAssuranceById1	130	testFindAssuranceById1
131	testFindAssuranceById2	131	testFindAssuranceById2
132	testFindAssuranceByOrderId	132	testFindAssuranceByOrderId
133	testFindAssuranceByOrderId1	133	testFindAssuranceByOrderId1
134	testFindAssuranceByOrderId2	134	testFindAssuranceByOrderId2
135	testFindByAccountId	135	testFindByAccountId
136	testFindByConsignee	136	testFindByConsignee
137	testFindByOrderId	137	testFindByOrderId
138	testFindByOrderId1	138	testFindByOrderId1
139	testFindByOrderId2	139	testFindByOrderId2
140	testFindByRouteIdAndTrainType1	140	testFindByRouteIdAndTrainType1
141	testFindByRouteIdAndTrainType2	141	testFindByRouteIdAndTrainType2
142	testFindByUserId1	142	testFindByUserId1
143	testFindByUserId2	143	testFindByUserId2
144	testFindByUserName1	144	testFindByUserName1
145	testFindByUserName2	145	testFindByUserName2
146	testFindContactsByAccountId	146	testFindContactsByAccountId
147	testFindContactsById1	147	testFindContactsById1
148	testFindContactsById2	148	testFindContactsById2
149	testFindFoodOrderByOrderId	149	testFindFoodOrderByOrderId
150	testFindOrderById1	150	testFindOrderById1
151	testFindOrderById2	151	testFindOrderById2
152	testGetAccount	152	testGetAccount
153	testGetAllAssuranceType	153	testGetAllAssuranceType
154	testGetAllAssuranceTypes	154	testGetAllAssuranceTypes
155	testGetAllAssurances	155	testGetAllAssurances
156	testGetAllAssurances1	156	testGetAllAssurances1
157	testGetAllAssurances2	157	testGetAllAssurances2
158	testGetAllConfigs	158	testGetAllConfigs
159	testGetAllContacts	159	testGetAllContacts
160	testGetAllContacts1	160	testGetAllContacts1
161	testGetAllContacts2	161	testGetAllContacts2
162	testGetAllFood	162	testGetAllFood
163	testGetAllFoodStores	163	testGetAllFoodStores
164	testGetAllOrders	164	testGetAllOrders
165	testGetAllOrders1	165	testGetAllOrders1
166	testGetAllOrders2	166	testGetAllOrders2
167	testGetAllPrices	167	testGetAllPrices
168	testGetAllRoutes	168	testGetAllRoutes
169	testGetAllRoutes1	169	testGetAllRoutes1
170	testGetAllRoutes2	170	testGetAllRoutes2
171	testGetAllStations	171	testGetAllStations
172	testGetAllTrainFood	172	testGetAllTrainFood
173	testGetAllTrains	173	testGetAllTrains
174	testGetAllTravels	174	testGetAllTravels
175	testGetAllTravels1	175	testGetAllTravels1
176	testGetAllTravels2	176	testGetAllTravels2
177	testGetAllUser	177	testGetAllUser
178	testGetAllUsers	178	testGetAllUsers
179	testGetAllUsers1	179	testGetAllUsers1
180	testGetAllUsers2	180	testGetAllUsers2
181	testGetAssuranceById	181	testGetAssuranceById
182	testGetByCheapest	182	testGetByCheapest
183	testGetByMinStation	183	testGetByMinStation
184	testGetByQuickest	184	testGetByQuickest
185	testGetCheapest	185	testGetCheapest
186	testGetCheapestRoutes	186	testGetCheapestRoutes
187	testGetContactsByContactsId	187	testGetContactsByContactsId
188	testGetFoodStoresByStationIds	188	testGetFoodStoresByStationIds
189	testGetFoodStoresByStationIds1	189	testGetFoodStoresByStationIds1
190	testGetFoodStoresByStationIds2	190	testGetFoodStoresByStationIds2
191	testGetFoodStoresOfStation	191	testGetFoodStoresOfStation
192	testGetHello	192	testGetHello
193	testGetImageCode	193	testGetImageCode
194	testGetLeftTicketOfInterva2	194	testGetLeftTicketOfInterva2
195	testGetLeftTicketOfInterval	195	testGetLeftTicketOfInterval
196	testGetMinStation	196	testGetMinStation
197	testGetMinStopStations	197	testGetMinStopStations
198	testGetOrderById	198	testGetOrderById
199	testGetOrderById1	199	testGetOrderById1
200	testGetOrderById2	200	testGetOrderById2
201	testGetOrderPrice	201	testGetOrderPrice
202	testGetOrderPrice1	202	testGetOrderPrice1
203	testGetOrderPrice2	203	testGetOrderPrice2
204	testGetPriceByWeightAndRegion	204	testGetPriceByWeightAndRegion
205	testGetPriceByWeightAndRegion1	205	testGetPriceByWeightAndRegion1
206	testGetPriceByWeightAndRegion2	206	testGetPriceByWeightAndRegion2
207	testGetPriceByWeightAndRegion3	207	testGetPriceByWeightAndRegion3
208	testGetPriceConfig	208	testGetPriceConfig
209	testGetPriceInfo	209	testGetPriceInfo
210	testGetQuickest	210	testGetQuickest
211	testGetQuickestRoutes	211	testGetQuickestRoutes
212	testGetRouteById1	212	testGetRouteById1
213	testGetRouteById2	213	testGetRouteById2
214	testGetRouteByStartAndTerminal1	214	testGetRouteByStartAndTerminal1
215	testGetRouteByStartAndTerminal2	215	testGetRouteByStartAndTerminal2
216	testGetRouteByTripId	216	testGetRouteByTripId
217	testGetRouteByTripId1	217	testGetRouteByTripId1
218	testGetRouteByTripId2	218	testGetRouteByTripId2
219	testGetSoldTickets1	219	testGetSoldTickets1
220	testGetSoldTickets2	220	testGetSoldTickets2
221	testGetTicketListByDateAndTripId	221	testGetTicketListByDateAndTripId

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
222	testGetToken	222	testGetToken
223	testGetToken1	223	testGetToken1
224	testGetToken2	224	testGetToken2
225	testGetTrainFoodOfTrip	225	testGetTrainFoodOfTrip
226	testGetTrainTypeByTripId	226	testGetTrainTypeByTripId
227	testGetTransferResult	227	testGetTransferResult
228	testGetTransferSearch	228	testGetTransferSearch
229	testGetTripAllDetailInfo	229	testGetTripAllDetailInfo
230	testGetTripByRoute1	230	testGetTripByRoute1
231	testGetTripByRoute2	231	testGetTripByRoute2
232	testGetTripsByRouteId	232	testGetTripsByRouteId
233	testGetUserByUserId	233	testGetUserByUserId
234	testGetUserByUserName	234	testGetUserByUserName
235	testHome	235	testHome
236	testImageCode	236	testImageCode
237	testInitOrder1	237	testInitOrder1
238	testInitOrder2	238	testInitOrder2
239	testInitPayment1	239	testInitPayment1
240	testInitPayment2	240	testInitPayment2
241	testInsertConsign	241	testInsertConsign
242	testInsertConsignRecord	242	testInsertConsignRecord
243	testListFoodStores1	243	testListFoodStores1
244	testListFoodStores2	244	testListFoodStores2
245	testListFoodStoresByStationId1	245	testListFoodStoresByStationId1
246	testListFoodStoresByStationId2	246	testListFoodStoresByStationId2
247	testListTrainFood1	247	testListTrainFood1
248	testListTrainFood2	248	testListTrainFood2
249	testListTrainFoodByTripId1	249	testListTrainFoodByTripId1
250	testListTrainFoodByTripId2	250	testListTrainFoodByTripId2
251	testModify1	251	testModify1
252	testModify2	252	testModify2
253	testModify3	253	testModify3
254	testModifyAssurance	254	testModifyAssurance
255	testModifyConfig	255	testModifyConfig
256	testModifyContact	256	testModifyContact
257	testModifyContacts	257	testModifyContacts
258	testModifyOrder	258	testModifyOrder
259	testModifyOrder1	259	testModifyOrder1
260	testModifyOrder2	260	testModifyOrder2
261	testModifyPrice	261	testModifyPrice
262	testModifyPriceConfig	262	testModifyPriceConfig
263	testModifySecurityConfig1	263	testModifySecurityConfig1
264	testModifySecurityConfig2	264	testModifySecurityConfig2
265	testModifyStation	265	testModifyStation
266	testModifyTrain	266	testModifyTrain
267	testOrderCancelSuccess	267	testOrderCancelSuccess
268	testOrderCancelSuccess1	268	testOrderCancelSuccess1
269	testOrderCancelSuccess2	269	testOrderCancelSuccess2
270	testOrderChangedSuccess	270	testOrderChangedSuccess
271	testOrderChangedSuccess1	271	testOrderChangedSuccess1
272	testOrderChangedSuccess2	272	testOrderChangedSuccess2
273	testOrderCreateSuccess	273	testOrderCreateSuccess
274	testOrderCreateSuccess1	274	testOrderCreateSuccess1
275	testOrderCreateSuccess2	275	testOrderCreateSuccess2
276	testPay	276	testPay
277	testPay1	277	testPay1
278	testPay2	278	testPay2
279	testPayDifference	279	testPayDifference
280	testPayOrder	280	testPayOrder
281	testPayOrder1	281	testPayOrder1
282	testPayOrder2	282	testPayOrder2
283	testPreserve	283	testPreserve
284	testPreserveSuccess	284	testPreserveSuccess
285	testPreserveSuccess1	285	testPreserveSuccess1
286	testPreserveSuccess2	286	testPreserveSuccess2
287	testQuery	287	testQuery
288	testQuery1	288	testQuery1
289	testQuery2	289	testQuery2
290	testQueryAccount	290	testQueryAccount
291	testQueryAddMoney	291	testQueryAddMoney
292	testQueryAddMoney1	292	testQueryAddMoney1
293	testQueryAddMoney2	293	testQueryAddMoney2
294	testQueryAll	294	testQueryAll
295	testQueryAll1	295	testQueryAll1
296	testQueryAll2	296	testQueryAll2
297	testQueryAlreadySoldOrders	297	testQueryAlreadySoldOrders
298	testQueryByAccountId1	298	testQueryByAccountId1
299	testQueryByAccountId2	299	testQueryByAccountId2
300	testQueryByConsignee1	300	testQueryByConsignee1
301	testQueryByConsignee2	301	testQueryByConsignee2
302	testQueryById	302	testQueryById
303	testQueryById1	303	testQueryById1
304	testQueryById2	304	testQueryById2
305	testQueryByIdBatch1	305	testQueryByIdBatch1
306	testQueryByIdBatch2	306	testQueryByIdBatch2
307	testQueryByOrderId1	307	testQueryByOrderId1
308	testQueryByOrderId2	308	testQueryByOrderId2
309	testQueryByStartAndTerminal	309	testQueryByStartAndTerminal
310	testQueryForId1	310	testQueryForId1
311	testQueryForId2	311	testQueryForId2
312	testQueryForIdBatch	312	testQueryForIdBatch
313	testQueryForIdBatch1	313	testQueryForIdBatch1
314	testQueryForIdBatch2	314	testQueryForIdBatch2
315	testQueryForNameBatch	315	testQueryForNameBatch
316	testQueryForStationId	316	testQueryForStationId
317	testQueryForTravel	317	testQueryForTravel
318	testQueryInfo1	318	testQueryInfo1
319	testQueryInfo2	319	testQueryInfo2
320	testQueryOrders	320	testQueryOrders
321	testQueryOrdersForRefresh	321	testQueryOrdersForRefresh
322	testQueryPayment	322	testQueryPayment
323	testQueryPayment1	323	testQueryPayment1
324	testQueryPayment2	324	testQueryPayment2
325	testQueryPriceInformation	325	testQueryPriceInformation
326	testQueryTrainType	326	testQueryTrainType
327	testRebook	327	testRebook
328	testRegisterUser	328	testRegisterUser
329	testRetrieve	329	testRetrieve
330	testRetrieve1	330	testRetrieve1
331	testRetrieve2	331	testRetrieve2

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
332	testSaveChanges1	332	testSaveChanges1
333	testSaveChanges2	333	testSaveChanges2
334	testSaveOrderInfo	334	testSaveOrderInfo
335	testSaveUser	335	testSaveUser
336	testSearchCheapestResult	336	testSearchCheapestResult
337	testSearchMinStopStations	337	testSearchMinStopStations
338	testSearchQuickestResult	338	testSearchQuickestResult
339	testTicketCollect1	339	testTicketCollect1
340	testTicketCollect2	340	testTicketCollect2
341	testTicketExecute1	341	testTicketExecute1
342	testTicketExecute2	342	testTicketExecute2
343	testUpdate	343	testUpdate
344	testUpdate1	344	testUpdate1
345	testUpdate2	345	testUpdate2
346	testUpdateConfig	346	testUpdateConfig
347	testUpdateConsign	347	testUpdateConsign
348	testUpdateConsignRecord1	348	testUpdateConsignRecord1
349	testUpdateConsignRecord2	349	testUpdateConsignRecord2
350	testUpdateFoodOrder	350	testUpdateFoodOrder
351	testUpdateFoodOrder1	351	testUpdateFoodOrder1
352	testUpdateFoodOrder2	352	testUpdateFoodOrder2
353	testUpdateOrder	353	testUpdateOrder
354	testUpdateOrder1	354	testUpdateOrder1
355	testUpdateOrder2	355	testUpdateOrder2
356	testUpdatePriceConfig1	356	testUpdatePriceConfig1
357	testUpdatePriceConfig2	357	testUpdatePriceConfig2
358	testUpdateTravel	358	testUpdateTravel
359	testUpdateTravel1	359	testUpdateTravel1
360	testUpdateTravel2	360	testUpdateTravel2
361	testUpdateTrip	361	testUpdateTrip
362	testUpdateUser	362	testUpdateUser
363	testUpdateUser1	363	testUpdateUser1
364	testUpdateUser2	364	testUpdateUser2
365	testVerifyCode	365	testVerifyCode

Table A.10: *Domain*: `invokes`; *Criterion*: `nonemptyEqSeq`. Refers to Figure A.54.

TP2		TP1	
TP Number	**TP Name**	**TP Number**	**TP Name**
1	tesCreate2	1	tesCreate2
2	testAddTravel1	2	testAddTravel1
3	testAddTravel2	3	testAddTravel2
4	testAddTravel3	4	testAddTravel3
5	testAddTravel4	5	testAddTravel4
6	testCreate1	6	testCreate1
7	testDelete1	7	testDelete1
8	testDelete2	8	testDelete2
9	testDeleteById1	9	testDeleteById1
10	testDeleteById2	10	testDeleteById2
11	testDeleteByOrderId1	11	testDeleteByOrderId1
12	testDeleteByOrderId2	12	testDeleteByOrderId2
13	testDeleteFoodOrder1	13	testDeleteFoodOrder1
14	testDeleteFoodOrder2	14	testDeleteFoodOrder2
15	testDeleteOrder1	15	testDeleteOrder1
16	testDeleteOrder2	16	testDeleteOrder2
17	testDeleteRoute1	17	testDeleteRoute1
18	testDeleteRoute2	18	testDeleteRoute2
19	testDeleteTravel1	19	testDeleteTravel1
20	testDeleteTravel2	20	testDeleteTravel2
21	testExist1	21	testExist1
22	testExist2	22	testExist2
23	testFindAllFoodOrder1	23	testFindAllFoodOrder1
24	testFindAllFoodOrder2	24	testFindAllFoodOrder2
25	testFindAllPriceConfig1	25	testFindAllPriceConfig1
26	testFindAllPriceConfig2	26	testFindAllPriceConfig2
27	testFindAllSecurityConfig1	27	testFindAllSecurityConfig1
28	testFindAllSecurityConfig2	28	testFindAllSecurityConfig2
29	testGetAllContacts1	29	testGetAllContacts1
30	testGetAllContacts2	30	testGetAllContacts2
31	testGetAllOrders1	31	testGetAllOrders1
32	testGetAllOrders2	32	testGetAllOrders2
33	testGetAllRoutes1	33	testGetAllRoutes1
34	testGetAllRoutes2	34	testGetAllRoutes2
35	testGetAllUsers1	35	testGetAllUsers1
36	testGetAllUsers2	36	testGetAllUsers2
37	testGetFoodStoresByStationIds1	37	testGetFoodStoresByStationIds1
38	testGetFoodStoresByStationIds2	38	testGetFoodStoresByStationIds2
39	testListFoodStores1	39	testListFoodStores1
40	testListFoodStores2	40	testListFoodStores2
41	testListFoodStoresByStationId1	41	testListFoodStoresByStationId1
42	testListFoodStoresByStationId2	42	testListFoodStoresByStationId2
43	testListTrainFood1	43	testListTrainFood1
44	testListTrainFood2	44	testListTrainFood2
45	testListTrainFoodByTripId1	45	testListTrainFoodByTripId1
46	testListTrainFoodByTripId2	46	testListTrainFoodByTripId2
47	testOrderCancelSuccess1	47	testOrderCancelSuccess1
48	testOrderCancelSuccess2	48	testOrderCancelSuccess2
49	testOrderChangedSuccess1	49	testOrderChangedSuccess1
50	testOrderChangedSuccess2	50	testOrderChangedSuccess2
51	testOrderCreateSuccess1	51	testOrderCreateSuccess1
52	testOrderCreateSuccess2	52	testOrderCreateSuccess2
53	testPreserveSuccess1	53	testPreserveSuccess1
54	testPreserveSuccess2	54	testPreserveSuccess2
55	testQuery1	55	testQuery1
56	testQuery2	56	testQuery2
57	testQueryAddMoney1	57	testQueryAddMoney1
58	testQueryAddMoney2	58	testQueryAddMoney2
59	testQueryAll1	59	testQueryAll1
60	testQueryAll2	60	testQueryAll2
61	testQueryByAccountId1	61	testQueryByAccountId1
62	testQueryByAccountId2	62	testQueryByAccountId2
63	testQueryByConsignee1	63	testQueryByConsignee1
64	testQueryByConsignee2	64	testQueryByConsignee2
65	testQueryPayment1	65	testQueryPayment1
66	testQueryPayment2	66	testQueryPayment2

	TP2		TP1
TP Number	TP Name	TP Number	TP Name
67	testRetrieve1	67	testRetrieve1
68	testRetrieve2	68	testRetrieve2
69	testUpdate1	69	testUpdate1
70	testUpdate2	70	testUpdate2

Table A.11: *Domain*: `invokes`; *Criterion*: `nonemptyEqSet`. Refers to Figure A.55.

	TP2		TP1
TP Number	TP Name	TP Number	TP Name
1	tesCreate2	1	tesCreate2
2	testAddOrder1	2	testAddOrder1
3	testAddOrder2	3	testAddOrder2
4	testAddTravel1	4	testAddTravel1
5	testAddTravel2	5	testAddTravel2
6	testAddTravel3	6	testAddTravel3
7	testAddTravel4	7	testAddTravel4
8	testCreate1	8	testCreate1
9	testCreate2	9	testCreate2
10	testCreateAndModify2	10	testCreateAndModify2
11	testCreateAndModify3	11	testCreateAndModify3
12	testCreateTrainFood1	12	testCreateTrainFood1
13	testCreateTrainFood2	13	testCreateTrainFood2
14	testDelete1	14	testDelete1
15	testDelete2	15	testDelete2
16	testDeleteById1	16	testDeleteById1
17	testDeleteById2	17	testDeleteById2
18	testDeleteByOrderId1	18	testDeleteByOrderId1
19	testDeleteByOrderId2	19	testDeleteByOrderId2
20	testDeleteFoodOrder1	20	testDeleteFoodOrder1
21	testDeleteFoodOrder2	21	testDeleteFoodOrder2
22	testDeleteOrder1	22	testDeleteOrder1
23	testDeleteOrder2	23	testDeleteOrder2
24	testDeleteRoute1	24	testDeleteRoute1
25	testDeleteRoute2	25	testDeleteRoute2
26	testDeleteTravel1	26	testDeleteTravel1
27	testDeleteTravel2	27	testDeleteTravel2
28	testExist1	28	testExist1
29	testExist2	29	testExist2
30	testFindAllFoodOrder1	30	testFindAllFoodOrder1
31	testFindAllFoodOrder2	31	testFindAllFoodOrder2
32	testFindAllPriceConfig1	32	testFindAllPriceConfig1
33	testFindAllPriceConfig2	33	testFindAllPriceConfig2
34	testFindAllSecurityConfig1	34	testFindAllSecurityConfig1
35	testFindAllSecurityConfig2	35	testFindAllSecurityConfig2
36	testGetAllContacts1	36	testGetAllContacts1
37	testGetAllContacts2	37	testGetAllContacts2
38	testGetAllOrders1	38	testGetAllOrders1
39	testGetAllOrders2	39	testGetAllOrders2
40	testGetAllRoutes1	40	testGetAllRoutes1
41	testGetAllRoutes2	41	testGetAllRoutes2
42	testGetAllUsers1	42	testGetAllUsers1
43	testGetAllUsers2	43	testGetAllUsers2
44	testGetFoodStoresByStationIds1	44	testGetFoodStoresByStationIds1
45	testGetFoodStoresByStationIds2	45	testGetFoodStoresByStationIds2
46	testInitOrder1	46	testInitOrder1
47	testInitOrder2	47	testInitOrder2
48	testInitPayment1	48	testInitPayment1
49	testInitPayment2	49	testInitPayment2
50	testListFoodStores1	50	testListFoodStores1
51	testListFoodStores2	51	testListFoodStores2
52	testListFoodStoresByStationId1	52	testListFoodStoresByStationId1
53	testListFoodStoresByStationId2	53	testListFoodStoresByStationId2
54	testListTrainFood1	54	testListTrainFood1
55	testListTrainFood2	55	testListTrainFood2
56	testListTrainFoodByTripId1	56	testListTrainFoodByTripId1
57	testListTrainFoodByTripId2	57	testListTrainFoodByTripId2
58	testOrderCancelSuccess1	58	testOrderCancelSuccess1
59	testOrderCancelSuccess2	59	testOrderCancelSuccess2
60	testOrderChangedSuccess1	60	testOrderChangedSuccess1
61	testOrderChangedSuccess2	61	testOrderChangedSuccess2
62	testOrderCreateSuccess1	62	testOrderCreateSuccess1
63	testOrderCreateSuccess2	63	testOrderCreateSuccess2
64	testPreserveSuccess1	64	testPreserveSuccess1
65	testPreserveSuccess2	65	testPreserveSuccess2
66	testQuery1	66	testQuery1
67	testQuery2	67	testQuery2
68	testQueryAddMoney1	68	testQueryAddMoney1
69	testQueryAddMoney2	69	testQueryAddMoney2
70	testQueryAll1	70	testQueryAll1
71	testQueryAll2	71	testQueryAll2
72	testQueryByAccountId1	72	testQueryByAccountId1
73	testQueryByAccountId2	73	testQueryByAccountId2
74	testQueryByConsignee1	74	testQueryByConsignee1
75	testQueryByConsignee2	75	testQueryByConsignee2
76	testQueryPayment1	76	testQueryPayment1
77	testQueryPayment2	77	testQueryPayment2
78	testRetrieve1	78	testRetrieve1
79	testRetrieve2	79	testRetrieve2
80	testUpdate1	80	testUpdate1
81	testUpdate2	81	testUpdate2
82	testUpdateOrder1	82	testUpdateOrder1
83	testUpdateOrder2	83	testUpdateOrder2
84	testUpdateTravel1	84	testUpdateTravel1
85	testUpdateTravel2	85	testUpdateTravel2

Table A.12: *Domain*: `invokes`; *Criterion*: `nonemptyIntersection`. Refers to Figure A.56.

	TP2		TP1
TP Number	TP Name	TP Number	TP Name
1	tesCreate2	1	tesCreate2

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
2	testAddConfig	2	testAddConfig
3	testAddContact	3	testAddContact
4	testAddContacts	4	testAddContacts
5	testAddCreateNewOrder	5	testAddCreateNewOrder
6	testAddMoney	6	testAddMoney
7	testAddMoney1	7	testAddMoney1
8	testAddMoney2	8	testAddMoney2
9	testAddNewOrder1	9	testAddNewOrder1
10	testAddNewOrder2	10	testAddNewOrder2
11	testAddNewSecurityConfig1	11	testAddNewSecurityConfig1
12	testAddNewSecurityConfig2	12	testAddNewSecurityConfig2
13	testAddOrder	13	testAddOrder
14	testAddOrder1	14	testAddOrder1
15	testAddOrder2	15	testAddOrder2
16	testAddPrice	16	testAddPrice
17	testAddRoute	17	testAddRoute
18	testAddStation	18	testAddStation
19	testAddTrain	19	testAddTrain
20	testAddTravel	20	testAddTravel
21	testAddTravel1	21	testAddTravel1
22	testAddTravel2	22	testAddTravel2
23	testAddTravel3	23	testAddTravel3
24	testAddTravel4	24	testAddTravel4
25	testAddUser	25	testAddUser
26	testAdminQueryAll	26	testAdminQueryAll
27	testAdminQueryAll1	27	testAdminQueryAll1
28	testAdminQueryAll2	28	testAdminQueryAll2
29	testAlterOrder1	29	testAlterOrder1
30	testAlterOrder2	30	testAlterOrder2
31	testCalculate	31	testCalculate
32	testCalculateRefund1	32	testCalculateRefund1
33	testCalculateRefund2	33	testCalculateRefund2
34	testCancelOrder1	34	testCancelOrder1
35	testCancelOrder2	35	testCancelOrder2
36	testCancelTicket	36	testCancelTicket
37	testCheck	37	testCheck
38	testCheckSecurityAboutOrder	38	testCheckSecurityAboutOrder
39	testCheckStationExists	39	testCheckStationExists
40	testCollectTicket	40	testCollectTicket
41	testCreate	41	testCreate
42	testCreate1	42	testCreate1
43	testCreate2	43	testCreate2
44	testCreate3	44	testCreate3
45	testCreateAccount	45	testCreateAccount
46	testCreateAccount1	46	testCreateAccount1
47	testCreateAccount2	47	testCreateAccount2
48	testCreateAndModify1	48	testCreateAndModify1
49	testCreateAndModify2	49	testCreateAndModify2
50	testCreateAndModify3	50	testCreateAndModify3
51	testCreateAndModifyPrice1	51	testCreateAndModifyPrice1
52	testCreateAndModifyPrice2	52	testCreateAndModifyPrice2
53	testCreateAndModifyRoute	53	testCreateAndModifyRoute
54	testCreateConfig	54	testCreateConfig
55	testCreateContacts1	55	testCreateContacts1
56	testCreateContacts2	56	testCreateContacts2
57	testCreateDefaultAuthUser	57	testCreateDefaultAuthUser
58	testCreateDefaultUser	58	testCreateDefaultUser
59	testCreateFoodOrder	59	testCreateFoodOrder
60	testCreateFoodOrder1	60	testCreateFoodOrder1
61	testCreateFoodOrder2	61	testCreateFoodOrder2
62	testCreateFoodStore1	62	testCreateFoodStore1
63	testCreateFoodStore2	63	testCreateFoodStore2
64	testCreateNewAssurance	64	testCreateNewAssurance
65	testCreateNewContacts	65	testCreateNewContacts
66	testCreateNewContactsAdmin	66	testCreateNewContactsAdmin
67	testCreateNewOrder	67	testCreateNewOrder
68	testCreateNewPriceConfig1	68	testCreateNewPriceConfig1
69	testCreateNewPriceConfig2	69	testCreateNewPriceConfig2
70	testCreateTrainFood1	70	testCreateTrainFood1
71	testCreateTrainFood2	71	testCreateTrainFood2
72	testCreateTrip	72	testCreateTrip
73	testDelete	73	testDelete
74	testDelete1	74	testDelete1
75	testDelete2	75	testDelete2
76	testDeleteAssurance	76	testDeleteAssurance
77	testDeleteAssuranceByOrderId	77	testDeleteAssuranceByOrderId
78	testDeleteById1	78	testDeleteById1
79	testDeleteById2	79	testDeleteById2
80	testDeleteByOrderId1	80	testDeleteByOrderId1
81	testDeleteByOrderId2	81	testDeleteByOrderId2
82	testDeleteByUserId	82	testDeleteByUserId
83	testDeleteConfig	83	testDeleteConfig
84	testDeleteContact	84	testDeleteContact
85	testDeleteContacts	85	testDeleteContacts
86	testDeleteFoodOrder	86	testDeleteFoodOrder
87	testDeleteFoodOrder1	87	testDeleteFoodOrder1
88	testDeleteFoodOrder2	88	testDeleteFoodOrder2
89	testDeleteOrder	89	testDeleteOrder
90	testDeleteOrder1	90	testDeleteOrder1
91	testDeleteOrder2	91	testDeleteOrder2
92	testDeletePrice	92	testDeletePrice
93	testDeletePriceConfig1	93	testDeletePriceConfig1
94	testDeletePriceConfig2	94	testDeletePriceConfig2
95	testDeleteRoute	95	testDeleteRoute
96	testDeleteRoute1	96	testDeleteRoute1
97	testDeleteRoute2	97	testDeleteRoute2
98	testDeleteSecurityConfig1	98	testDeleteSecurityConfig1
99	testDeleteSecurityConfig2	99	testDeleteSecurityConfig2
100	testDeleteStation	100	testDeleteStation
101	testDeleteTrain	101	testDeleteTrain
102	testDeleteTravel	102	testDeleteTravel
103	testDeleteTravel1	103	testDeleteTravel1
104	testDeleteTravel2	104	testDeleteTravel2
105	testDeleteTrip	105	testDeleteTrip
106	testDeleteUser	106	testDeleteUser
107	testDeleteUser1	107	testDeleteUser1
108	testDeleteUser2	108	testDeleteUser2
109	testDeleteUserAuth	109	testDeleteUserAuth
110	testDeleteUserById	110	testDeleteUserById
111	testDipatchSeat	111	testDipatchSeat

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
112	testDistributeSeat1	112	testDistributeSeat1
113	testDistributeSeat2	113	testDistributeSeat2
114	testDrawBack	114	testDrawBack
115	testDrawBack1	115	testDrawBack1
116	testDrawBack2	116	testDrawBack2
117	testDrawbackMoney	117	testDrawbackMoney
118	testExecuteTicket	118	testExecuteTicket
119	testExist1	119	testExist1
120	testExist2	120	testExist2
121	testFindAllFoodOrder	121	testFindAllFoodOrder
122	testFindAllFoodOrder1	122	testFindAllFoodOrder1
123	testFindAllFoodOrder2	123	testFindAllFoodOrder2
124	testFindAllOrder	124	testFindAllOrder
125	testFindAllPriceConfig1	125	testFindAllPriceConfig1
126	testFindAllPriceConfig2	126	testFindAllPriceConfig2
127	testFindAllSecurityConfig	127	testFindAllSecurityConfig
128	testFindAllSecurityConfig1	128	testFindAllSecurityConfig1
129	testFindAllSecurityConfig2	129	testFindAllSecurityConfig2
130	testFindAssuranceById1	130	testFindAssuranceById1
131	testFindAssuranceById2	131	testFindAssuranceById2
132	testFindAssuranceByOrderId	132	testFindAssuranceByOrderId
133	testFindAssuranceByOrderId1	133	testFindAssuranceByOrderId1
134	testFindAssuranceByOrderId2	134	testFindAssuranceByOrderId2
135	testFindByAccountId	135	testFindByAccountId
136	testFindByConsignee	136	testFindByConsignee
137	testFindByOrderId	137	testFindByOrderId
138	testFindByOrderId1	138	testFindByOrderId1
139	testFindByOrderId2	139	testFindByOrderId2
140	testFindByRouteIdAndTrainType1	140	testFindByRouteIdAndTrainType1
141	testFindByRouteIdAndTrainType2	141	testFindByRouteIdAndTrainType2
142	testFindByUserId1	142	testFindByUserId1
143	testFindByUserId2	143	testFindByUserId2
144	testFindByUserName1	144	testFindByUserName1
145	testFindByUserName2	145	testFindByUserName2
146	testFindContactsByAccountId	146	testFindContactsByAccountId
147	testFindContactsById1	147	testFindContactsById1
148	testFindContactsById2	148	testFindContactsById2
149	testFindFoodOrderByOrderId	149	testFindFoodOrderByOrderId
150	testFindOrderById1	150	testFindOrderById1
151	testFindOrderById2	151	testFindOrderById2
152	testGetAccount	152	testGetAccount
153	testGetAllAssuranceType	153	testGetAllAssuranceType
154	testGetAllAssuranceTypes	154	testGetAllAssuranceTypes
155	testGetAllAssurances	155	testGetAllAssurances
156	testGetAllAssurances1	156	testGetAllAssurances1
157	testGetAllAssurances2	157	testGetAllAssurances2
158	testGetAllConfigs	158	testGetAllConfigs
159	testGetAllContacts	159	testGetAllContacts
160	testGetAllContacts1	160	testGetAllContacts1
161	testGetAllContacts2	161	testGetAllContacts2
162	testGetAllFood	162	testGetAllFood
163	testGetAllFoodStores	163	testGetAllFoodStores
164	testGetAllOrders	164	testGetAllOrders
165	testGetAllOrders1	165	testGetAllOrders1
166	testGetAllOrders2	166	testGetAllOrders2
167	testGetAllPrices	167	testGetAllPrices
168	testGetAllRoutes	168	testGetAllRoutes
169	testGetAllRoutes1	169	testGetAllRoutes1
170	testGetAllRoutes2	170	testGetAllRoutes2
171	testGetAllStations	171	testGetAllStations
172	testGetAllTrainFood	172	testGetAllTrainFood
173	testGetAllTrains	173	testGetAllTrains
174	testGetAllTravels	174	testGetAllTravels
175	testGetAllTravels1	175	testGetAllTravels1
176	testGetAllTravels2	176	testGetAllTravels2
177	testGetAllUser	177	testGetAllUser
178	testGetAllUsers	178	testGetAllUsers
179	testGetAllUsers1	179	testGetAllUsers1
180	testGetAllUsers2	180	testGetAllUsers2
181	testGetAssuranceById	181	testGetAssuranceById
182	testGetByCheapest	182	testGetByCheapest
183	testGetByMinStation	183	testGetByMinStation
184	testGetByQuickest	184	testGetByQuickest
185	testGetCheapest	185	testGetCheapest
186	testGetCheapestRoutes	186	testGetCheapestRoutes
187	testGetContactsByContactsId	187	testGetContactsByContactsId
188	testGetFoodStoresByStationIds	188	testGetFoodStoresByStationIds
189	testGetFoodStoresByStationIds1	189	testGetFoodStoresByStationIds1
190	testGetFoodStoresByStationIds2	190	testGetFoodStoresByStationIds2
191	testGetFoodStoresOfStation	191	testGetFoodStoresOfStation
192	testGetHello	192	testGetHello
193	testGetImageCode	193	testGetImageCode
194	testGetLeftTicketOfInterva2	194	testGetLeftTicketOfInterva2
195	testGetLeftTicketOfInterval	195	testGetLeftTicketOfInterval
196	testGetMinStation	196	testGetMinStation
197	testGetMinStopStations	197	testGetMinStopStations
198	testGetOrderById	198	testGetOrderById
199	testGetOrderById1	199	testGetOrderById1
200	testGetOrderById2	200	testGetOrderById2
201	testGetOrderPrice	201	testGetOrderPrice
202	testGetOrderPrice1	202	testGetOrderPrice1
203	testGetOrderPrice2	203	testGetOrderPrice2
204	testGetPriceByWeightAndRegion	204	testGetPriceByWeightAndRegion
205	testGetPriceByWeightAndRegion1	205	testGetPriceByWeightAndRegion1
206	testGetPriceByWeightAndRegion2	206	testGetPriceByWeightAndRegion2
207	testGetPriceByWeightAndRegion3	207	testGetPriceByWeightAndRegion3
208	testGetPriceConfig	208	testGetPriceConfig
209	testGetPriceInfo	209	testGetPriceInfo
210	testGetQuickest	210	testGetQuickest
211	testGetQuickestRoutes	211	testGetQuickestRoutes
212	testGetRouteById1	212	testGetRouteById1
213	testGetRouteById2	213	testGetRouteById2
214	testGetRouteByStartAndTerminal1	214	testGetRouteByStartAndTerminal1
215	testGetRouteByStartAndTerminal2	215	testGetRouteByStartAndTerminal2
216	testGetRouteByTripId	216	testGetRouteByTripId
217	testGetRouteByTripId1	217	testGetRouteByTripId1
218	testGetRouteByTripId2	218	testGetRouteByTripId2
219	testGetSoldTickets1	219	testGetSoldTickets1
220	testGetSoldTickets2	220	testGetSoldTickets2
221	testGetTicketListByDateAndTripId	221	testGetTicketListByDateAndTripId

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
222	testGetToken	222	testGetToken
223	testGetToken1	223	testGetToken1
224	testGetToken2	224	testGetToken2
225	testGetTrainFoodOfTrip	225	testGetTrainFoodOfTrip
226	testGetTrainTypeByTripId	226	testGetTrainTypeByTripId
227	testGetTransferResult	227	testGetTransferResult
228	testGetTransferSearch	228	testGetTransferSearch
229	testGetTripAllDetailInfo	229	testGetTripAllDetailInfo
230	testGetTripByRoute1	230	testGetTripByRoute1
231	testGetTripByRoute2	231	testGetTripByRoute2
232	testGetTripsByRouteId	232	testGetTripsByRouteId
233	testGetUserByUserId	233	testGetUserByUserId
234	testGetUserByUserName	234	testGetUserByUserName
235	testHome	235	testHome
236	testImageCode	236	testImageCode
237	testInitOrder1	237	testInitOrder1
238	testInitOrder2	238	testInitOrder2
239	testInitPayment1	239	testInitPayment1
240	testInitPayment2	240	testInitPayment2
241	testInsertConsign	241	testInsertConsign
242	testInsertConsignRecord	242	testInsertConsignRecord
243	testListFoodStores1	243	testListFoodStores1
244	testListFoodStores2	244	testListFoodStores2
245	testListFoodStoresByStationId1	245	testListFoodStoresByStationId1
246	testListFoodStoresByStationId2	246	testListFoodStoresByStationId2
247	testListTrainFood1	247	testListTrainFood1
248	testListTrainFood2	248	testListTrainFood2
249	testListTrainFoodByTripId1	249	testListTrainFoodByTripId1
250	testListTrainFoodByTripId2	250	testListTrainFoodByTripId2
251	testModify1	251	testModify1
252	testModify2	252	testModify2
253	testModify3	253	testModify3
254	testModifyAssurance	254	testModifyAssurance
255	testModifyConfig	255	testModifyConfig
256	testModifyContact	256	testModifyContact
257	testModifyContacts	257	testModifyContacts
258	testModifyOrder	258	testModifyOrder
259	testModifyOrder1	259	testModifyOrder1
260	testModifyOrder2	260	testModifyOrder2
261	testModifyPrice	261	testModifyPrice
262	testModifyPriceConfig	262	testModifyPriceConfig
263	testModifySecurityConfig1	263	testModifySecurityConfig1
264	testModifySecurityConfig2	264	testModifySecurityConfig2
265	testModifyStation	265	testModifyStation
266	testModifyTrain	266	testModifyTrain
267	testOrderCancelSuccess	267	testOrderCancelSuccess
268	testOrderCancelSuccess1	268	testOrderCancelSuccess1
269	testOrderCancelSuccess2	269	testOrderCancelSuccess2
270	testOrderChangedSuccess	270	testOrderChangedSuccess
271	testOrderChangedSuccess1	271	testOrderChangedSuccess1
272	testOrderChangedSuccess2	272	testOrderChangedSuccess2
273	testOrderCreateSuccess	273	testOrderCreateSuccess
274	testOrderCreateSuccess1	274	testOrderCreateSuccess1
275	testOrderCreateSuccess2	275	testOrderCreateSuccess2
276	testPay	276	testPay
277	testPay1	277	testPay1
278	testPay2	278	testPay2
279	testPayDifference	279	testPayDifference
280	testPayOrder	280	testPayOrder
281	testPayOrder1	281	testPayOrder1
282	testPayOrder2	282	testPayOrder2
283	testPreserve	283	testPreserve
284	testPreserveSuccess	284	testPreserveSuccess
285	testPreserveSuccess1	285	testPreserveSuccess1
286	testPreserveSuccess2	286	testPreserveSuccess2
287	testQuery	287	testQuery
288	testQuery1	288	testQuery1
289	testQuery2	289	testQuery2
290	testQueryAccount	290	testQueryAccount
291	testQueryAddMoney	291	testQueryAddMoney
292	testQueryAddMoney1	292	testQueryAddMoney1
293	testQueryAddMoney2	293	testQueryAddMoney2
294	testQueryAll	294	testQueryAll
295	testQueryAll1	295	testQueryAll1
296	testQueryAll2	296	testQueryAll2
297	testQueryAlreadySoldOrders	297	testQueryAlreadySoldOrders
298	testQueryByAccountId1	298	testQueryByAccountId1
299	testQueryByAccountId2	299	testQueryByAccountId2
300	testQueryByConsignee1	300	testQueryByConsignee1
301	testQueryByConsignee2	301	testQueryByConsignee2
302	testQueryById	302	testQueryById
303	testQueryById1	303	testQueryById1
304	testQueryById2	304	testQueryById2
305	testQueryByIdBatch1	305	testQueryByIdBatch1
306	testQueryByIdBatch2	306	testQueryByIdBatch2
307	testQueryByOrderId1	307	testQueryByOrderId1
308	testQueryByOrderId2	308	testQueryByOrderId2
309	testQueryByStartAndTerminal	309	testQueryByStartAndTerminal
310	testQueryForId1	310	testQueryForId1
311	testQueryForId2	311	testQueryForId2
312	testQueryForIdBatch	312	testQueryForIdBatch
313	testQueryForIdBatch1	313	testQueryForIdBatch1
314	testQueryForIdBatch2	314	testQueryForIdBatch2
315	testQueryForNameBatch	315	testQueryForNameBatch
316	testQueryForStationId	316	testQueryForStationId
317	testQueryForTravel	317	testQueryForTravel
318	testQueryInfo1	318	testQueryInfo1
319	testQueryInfo2	319	testQueryInfo2
320	testQueryOrders	320	testQueryOrders
321	testQueryOrdersForRefresh	321	testQueryOrdersForRefresh
322	testQueryPayment	322	testQueryPayment
323	testQueryPayment1	323	testQueryPayment1
324	testQueryPayment2	324	testQueryPayment2
325	testQueryPriceInformation	325	testQueryPriceInformation
326	testQueryTrainType	326	testQueryTrainType
327	testRebook	327	testRebook
328	testRegisterUser	328	testRegisterUser
329	testRetrieve	329	testRetrieve
330	testRetrieve1	330	testRetrieve1
331	testRetrieve2	331	testRetrieve2

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
332	testSaveChanges1	332	testSaveChanges1
333	testSaveChanges2	333	testSaveChanges2
334	testSaveOrderInfo	334	testSaveOrderInfo
335	testSaveUser	335	testSaveUser
336	testSearchCheapestResult	336	testSearchCheapestResult
337	testSearchMinStopStations	337	testSearchMinStopStations
338	testSearchQuickestResult	338	testSearchQuickestResult
339	testTicketCollect1	339	testTicketCollect1
340	testTicketCollect2	340	testTicketCollect2
341	testTicketExecute1	341	testTicketExecute1
342	testTicketExecute2	342	testTicketExecute2
343	testUpdate	343	testUpdate
344	testUpdate1	344	testUpdate1
345	testUpdate2	345	testUpdate2
346	testUpdateConfig	346	testUpdateConfig
347	testUpdateConsign	347	testUpdateConsign
348	testUpdateConsignRecord1	348	testUpdateConsignRecord1
349	testUpdateConsignRecord2	349	testUpdateConsignRecord2
350	testUpdateFoodOrder	350	testUpdateFoodOrder
351	testUpdateFoodOrder1	351	testUpdateFoodOrder1
352	testUpdateFoodOrder2	352	testUpdateFoodOrder2
353	testUpdateOrder	353	testUpdateOrder
354	testUpdateOrder1	354	testUpdateOrder1
355	testUpdateOrder2	355	testUpdateOrder2
356	testUpdatePriceConfig1	356	testUpdatePriceConfig1
357	testUpdatePriceConfig2	357	testUpdatePriceConfig2
358	testUpdateTravel	358	testUpdateTravel
359	testUpdateTravel1	359	testUpdateTravel1
360	testUpdateTravel2	360	testUpdateTravel2
361	testUpdateTrip	361	testUpdateTrip
362	testUpdateUser	362	testUpdateUser
363	testUpdateUser1	363	testUpdateUser1
364	testUpdateUser2	364	testUpdateUser2
365	testVerifyCode	365	testVerifyCode

Table A.13: *Domain*: `invokes`; *Criterion*: `nonemptySubSeq`. Refers to Figure A.57.

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
1	tesCreate2	1	tesCreate2
2	testAddMoney1	2	testAddMoney2
3	testAddOrder2	3	testAddOrder1
4	testAddTravel1	4	testAddTravel1
5	testAddTravel2	5	testAddTravel2
6	testAddTravel3	6	testAddTravel3
7	testAddTravel4	7	testAddTravel4
8	testAlterOrder2	8	testCalculateRefund1
9	testCalculateRefund2	9	testCancelOrder1
10	testCancelOrder2	10	testCheckStationExists
11	testCreate1	11	testCreate1
12	testCreate2	12	testCreate2
13	testCreateAccount1	13	testCreate3
14	testCreateAndModify3	14	testCreateAccount2
15	testCreateAndModifyPrice2	15	testCreateAndModify2
16	testCreateContacts1	16	testCreateAndModifyPrice1
17	testCreateNewPriceConfig2	17	testCreateContacts2
18	testCreateTrainFood1	18	testCreateNewPriceConfig1
19	testDelete1	19	testCreateTrainFood2
20	testDelete2	20	testDelete1
21	testDeleteById1	21	testDelete2
22	testDeleteById2	22	testDeleteById1
23	testDeleteByOrderId1	23	testDeleteById2
24	testDeleteByOrderId2	24	testDeleteByOrderId1
25	testDeleteFoodOrder1	25	testDeleteByOrderId2
26	testDeleteFoodOrder2	26	testDeleteFoodOrder1
27	testDeleteOrder1	27	testDeleteFoodOrder2
28	testDeleteOrder2	28	testDeleteOrder1
29	testDeletePriceConfig2	29	testDeleteOrder2
30	testDeleteRoute1	30	testDeletePriceConfig1
31	testDeleteRoute2	31	testDeleteRoute1
32	testDeleteSecurityConfig1	32	testDeleteRoute2
33	testDeleteTravel1	33	testDeleteSecurityConfig2
34	testDeleteTravel2	34	testDeleteTravel1
35	testDeleteUser1	35	testDeleteTravel2
36	testDrawBack1	36	testDeleteUser2
37	testExist1	37	testDeleteUserAuth
38	testExist2	38	testDrawBack2
39	testFindAllFoodOrder1	39	testExist1
40	testFindAllFoodOrder2	40	testExist2
41	testFindAllPriceConfig1	41	testFindAllFoodOrder1
42	testFindAllPriceConfig2	42	testFindAllFoodOrder2
43	testFindAllSecurityConfig1	43	testFindAllPriceConfig1
44	testFindAllSecurityConfig2	44	testFindAllPriceConfig2
45	testFindAssuranceById2	45	testFindAllSecurityConfig1
46	testFindAssuranceByOrderId2	46	testFindAllSecurityConfig2
47	testFindByOrderId1	47	testFindAssuranceById1
48	testFindByRouteIdAndTrainType2	48	testFindAssuranceById2
49	testFindByUserId1	49	testFindAssuranceByOrderId1
50	testFindByUserName1	50	testFindByOrderId2
51	testFindContactsById1	51	testFindByRouteIdAndTrainType1
52	testFindOrderById2	52	testFindByUserId2
53	testGetAllAssurances1	53	testFindByUserName2
54	testGetAllContacts1	54	testFindContactsById1
55	testGetAllContacts2	55	testFindContactsById2
56	testGetAllOrders1	56	testFindOrderById1
57	testGetAllOrders2	57	testGetAccount
58	testGetAllRoutes1	58	testGetAllAssurances2
59	testGetAllRoutes2	59	testGetAllContacts1
60	testGetAllUsers1	60	testGetAllContacts2
61	testGetAllUsers2	61	testGetAllOrders1
62	testGetFoodStoresByStationIds1	62	testGetAllOrders2
63	testGetFoodStoresByStationIds2	63	testGetAllRoutes1
64	testGetOrderById2	64	testGetAllRoutes2
65	testGetOrderPrice2	65	testGetAllUsers1
66	testGetPriceByWeightAndRegion2	66	testGetAllUsers2

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
67	testGetPriceByWeightAndRegion3	67	testGetFoodStoresByStationIds1
68	testGetRouteById2	68	testGetFoodStoresByStationIds2
69	testGetRouteByStartAndTerminal1	69	testGetOrderById1
70	testGetTripByRoute1	70	testGetOrderPrice1
71	testInitOrder2	71	testGetPriceByWeightAndRegion1
72	testInitPayment2	72	testGetRouteById1
73	testListFoodStores1	73	testGetRouteByStartAndTerminal2
74	testListFoodStores2	74	testGetTripByRoute2
75	testListFoodStoresByStationId1	75	testInitOrder1
76	testListFoodStoresByStationId2	76	testInitPayment1
77	testListTrainFood1	77	testListFoodStores1
78	testListTrainFood2	78	testListFoodStores2
79	testListTrainFoodByTripId1	79	testListFoodStoresByStationId1
80	testListTrainFoodByTripId2	80	testListFoodStoresByStationId2
81	testModify1	81	testListTrainFood1
82	testModify2	82	testListTrainFood2
83	testModify3	83	testListTrainFoodByTripId1
84	testModifySecurityConfig2	84	testListTrainFoodByTripId2
85	testOrderCancelSuccess1	85	testModify1
86	testOrderCancelSuccess2	86	testModify3
87	testOrderChangedSuccess1	87	testModifySecurityConfig1
88	testOrderChangedSuccess2	88	testOrderCancelSuccess1
89	testOrderCreateSuccess1	89	testOrderCancelSuccess2
90	testOrderCreateSuccess2	90	testOrderChangedSuccess1
91	testPay1	91	testOrderChangedSuccess2
92	testPreserve	92	testOrderCreateSuccess1
93	testPreserveSuccess1	93	testOrderCreateSuccess2
94	testPreserveSuccess2	94	testPay2
95	testQuery1	95	testPreserveSuccess1
96	testQuery2	96	testPreserveSuccess2
97	testQueryAddMoney1	97	testQuery1
98	testQueryAddMoney2	98	testQuery2
99	testQueryAll1	99	testQueryAddMoney1
100	testQueryAll2	100	testQueryAddMoney2
101	testQueryByAccountId1	101	testQueryAll1
102	testQueryByAccountId2	102	testQueryAll2
103	testQueryByConsignee1	103	testQueryByAccountId1
104	testQueryByConsignee2	104	testQueryByAccountId2
105	testQueryById1	105	testQueryByConsignee1
106	testQueryByOrderId1	106	testQueryByConsignee2
107	testQueryForId1	107	testQueryById2
108	testQueryForIdBatch1	108	testQueryByOrderId2
109	testQueryForTravel	109	testQueryForId2
110	testQueryInfo2	110	testQueryForIdBatch2
111	testQueryOrdersForRefresh	111	testQueryForStationId
112	testQueryPayment1	112	testQueryInfo1
113	testQueryPayment2	113	testQueryPayment1
114	testRetrieve1	114	testQueryPayment2
115	testRetrieve2	115	testQueryTrainType
116	testSaveChanges2	116	testRetrieve1
117	testUpdate1	117	testRetrieve2
118	testUpdate2	118	testSaveChanges1
119	testUpdateFoodOrder2	119	testUpdate1
120	testUpdateOrder2	120	testUpdate2
121	testUpdatePriceConfig2	121	testUpdateFoodOrder1
122	testUpdateTravel2	122	testUpdateOrder1
		123	testUpdatePriceConfig1
		124	testUpdateTravel1

Table A.14: *Domain*: `invokes`; *Criterion*: `nonemptySubSet`. Refers to Figure A.58.

	TP2		TP1
TP Number	**TP Name**	**TP Number**	**TP Name**
1	tesCreate2	1	tesCreate2
2	testAddMoney1	2	testAddMoney2
3	testAddOrder1	3	testAddOrder1
4	testAddOrder2	4	testAddOrder2
5	testAddTravel1	5	testAddTravel1
6	testAddTravel2	6	testAddTravel2
7	testAddTravel3	7	testAddTravel3
8	testAddTravel4	8	testAddTravel4
9	testAlterOrder2	9	testCalculateRefund1
10	testCalculateRefund2	10	testCancelOrder1
11	testCancelOrder2	11	testCheckStationExists
12	testCreate1	12	testCreate1
13	testCreate2	13	testCreate2
14	testCreateAccount1	14	testCreate3
15	testCreateAndModify2	15	testCreateAccount2
16	testCreateAndModify3	16	testCreateAndModify2
17	testCreateAndModifyPrice2	17	testCreateAndModify3
18	testCreateContacts1	18	testCreateAndModifyPrice1
19	testCreateFoodStore2	19	testCreateContacts2
20	testCreateNewPriceConfig2	20	testCreateFoodStore1
21	testCreateTrainFood1	21	testCreateNewPriceConfig1
22	testCreateTrainFood2	22	testCreateTrainFood1
23	testDelete1	23	testCreateTrainFood2
24	testDelete2	24	testDelete1
25	testDeleteById1	25	testDelete2
26	testDeleteById2	26	testDeleteById1
27	testDeleteByOrderId1	27	testDeleteById2
28	testDeleteByOrderId2	28	testDeleteByOrderId1
29	testDeleteFoodOrder1	29	testDeleteByOrderId2
30	testDeleteFoodOrder2	30	testDeleteFoodOrder1
31	testDeleteOrder1	31	testDeleteFoodOrder2
32	testDeleteOrder2	32	testDeleteOrder1
33	testDeletePriceConfig2	33	testDeleteOrder2
34	testDeleteRoute1	34	testDeletePriceConfig1
35	testDeleteRoute2	35	testDeleteRoute1
36	testDeleteSecurityConfig1	36	testDeleteRoute2
37	testDeleteTravel1	37	testDeleteSecurityConfig2
38	testDeleteTravel2	38	testDeleteTravel1
39	testDeleteUser1	39	testDeleteTravel2
40	testDrawBack1	40	testDeleteUser2
41	testExist1	41	testDeleteUserAuth
42	testExist2	42	testDrawBack2

	TP2		TP1
TP Number	TP Name	TP Number	TP Name
43	testFindAllFoodOrder1	43	testExist1
44	testFindAllFoodOrder2	44	testExist2
45	testFindAllPriceConfig1	45	testFindAllFoodOrder1
46	testFindAllPriceConfig2	46	testFindAllFoodOrder2
47	testFindAllSecurityConfig1	47	testFindAllPriceConfig1
48	testFindAllSecurityConfig2	48	testFindAllPriceConfig2
49	testFindAssuranceById2	49	testFindAllSecurityConfig1
50	testFindAssuranceByOrderId2	50	testFindAllSecurityConfig2
51	testFindByOrderId1	51	testFindAssuranceById1
52	testFindByRouteIdAndTrainType2	52	testFindAssuranceById2
53	testFindByUserId1	53	testFindAssuranceByOrderId1
54	testFindByUserName1	54	testFindByOrderId2
55	testFindContactsById1	55	testFindByRouteIdAndTrainType1
56	testFindOrderById2	56	testFindByUserId2
57	testGetAllAssurances1	57	testFindByUserName2
58	testGetAllContacts1	58	testFindContactsById1
59	testGetAllContacts2	59	testFindContactsById2
60	testGetAllOrders1	60	testFindOrderById1
61	testGetAllOrders2	61	testGetAccount
62	testGetAllRoutes1	62	testGetAllAssurances2
63	testGetAllRoutes2	63	testGetAllContacts1
64	testGetAllTravels1	64	testGetAllContacts2
65	testGetAllUsers1	65	testGetAllOrders1
66	testGetAllUsers2	66	testGetAllOrders2
67	testGetFoodStoresByStationIds1	67	testGetAllRoutes1
68	testGetFoodStoresByStationIds2	68	testGetAllRoutes2
69	testGetOrderById2	69	testGetAllTravels2
70	testGetOrderPrice2	70	testGetAllUsers1
71	testGetPriceByWeightAndRegion2	71	testGetAllUsers2
72	testGetPriceByWeightAndRegion3	72	testGetFoodStoresByStationIds1
73	testGetRouteById2	73	testGetFoodStoresByStationIds2
74	testGetRouteByStartAndTerminal1	74	testGetOrderById1
75	testGetTripByRoute1	75	testGetOrderPrice1
76	testInitOrder1	76	testGetPriceByWeightAndRegion1
77	testInitOrder2	77	testGetRouteById1
78	testInitPayment1	78	testGetRouteByStartAndTerminal2
79	testInitPayment2	79	testGetTripByRoute2
80	testListFoodStores1	80	testInitOrder1
81	testListFoodStores2	81	testInitOrder2
82	testListFoodStoresByStationId1	82	testInitPayment1
83	testListFoodStoresByStationId2	83	testInitPayment2
84	testListTrainFood1	84	testListFoodStores1
85	testListTrainFood2	85	testListFoodStores2
86	testListTrainFoodByTripId1	86	testListFoodStoresByStationId1
87	testListTrainFoodByTripId2	87	testListFoodStoresByStationId2
88	testModify1	88	testListTrainFood1
89	testModify2	89	testListTrainFood2
90	testModify3	90	testListTrainFoodByTripId1
91	testModifySecurityConfig2	91	testListTrainFoodByTripId2
92	testOrderCancelSuccess1	92	testModify1
93	testOrderCancelSuccess2	93	testModify3
94	testOrderChangedSuccess1	94	testModifySecurityConfig1
95	testOrderChangedSuccess2	95	testOrderCancelSuccess1
96	testOrderCreateSuccess1	96	testOrderCancelSuccess2
97	testOrderCreateSuccess2	97	testOrderChangedSuccess1
98	testPay1	98	testOrderChangedSuccess2
99	testPreserve	99	testOrderCreateSuccess1
100	testPreserveSuccess1	100	testOrderCreateSuccess2
101	testPreserveSuccess2	101	testPay2
102	testQuery1	102	testPreserveSuccess1
103	testQuery2	103	testPreserveSuccess2
104	testQueryAddMoney1	104	testQuery1
105	testQueryAddMoney2	105	testQuery2
106	testQueryAll1	106	testQueryAddMoney1
107	testQueryAll2	107	testQueryAddMoney2
108	testQueryByAccountId1	108	testQueryAll1
109	testQueryByAccountId2	109	testQueryAll2
110	testQueryByConsignee1	110	testQueryByAccountId1
111	testQueryByConsignee2	111	testQueryByAccountId2
112	testQueryById1	112	testQueryByConsignee1
113	testQueryByOrderId1	113	testQueryByConsignee2
114	testQueryForId1	114	testQueryById2
115	testQueryForIdBatch1	115	testQueryByOrderId2
116	testQueryForTravel	116	testQueryForId2
117	testQueryInfo2	117	testQueryForIdBatch2
118	testQueryOrdersForRefresh	118	testQueryForStationId
119	testQueryPayment1	119	testQueryInfo1
120	testQueryPayment2	120	testQueryPayment1
121	testRetrieve1	121	testQueryPayment2
122	testRetrieve2	122	testQueryTrainType
123	testSaveChanges2	123	testRetrieve1
124	testUpdate1	124	testRetrieve2
125	testUpdate2	125	testSaveChanges1
126	testUpdateFoodOrder2	126	testUpdate1
127	testUpdateOrder1	127	testUpdate2
128	testUpdateOrder2	128	testUpdateFoodOrder1
129	testUpdatePriceConfig2	129	testUpdateOrder1
130	testUpdateTravel1	130	testUpdateOrder2
131	testUpdateTravel2	131	testUpdatePriceConfig1
		132	testUpdateTravel1
		133	testUpdateTravel2