ESEC/FSE: G: Automated Generation of Test Oracles for REST APIs

Juan C. Alonso (Supervisors: Sergio Segura, Antonio Ruiz-Cortés)
SCORE Lab, I3US Institute, Universidad de Sevilla, Seville, Spain

ABSTRACT

Test case generation tools for REST APIs have grown in number and complexity in recent years. However, their advanced capabilities for automated input generation contrast with the simplicity of their test oracles (i.e., mechanisms for determining whether a test has passed or failed), which limits the types of failures they can detect to crashes, regressions, and violations of the API specification or design best practices. In this paper, we present AGORA, an approach for the automated generation of test oracles for REST APIs through the detection of *invariants*—properties of the output that should always hold. In practice, AGORA aims to learn the expected behavior of an API by analyzing previous API requests and their corresponding responses. For this, we extended the Daikon tool for dynamic detection of likely invariants, including the definition of new types of invariants and the implementation of an *instrumenter* called Beet. Beet converts any OpenAPI specification and a collection of API requests and responses to a format processable by Daikon. As a result, AGORA currently supports the detection of up to 105 different types of invariants in REST APIs. AGORA achieved a total precision of 81.2% when tested on a dataset of 11 operations from 7 industrial APIs. More importantly, the test oracles generated by AGORA detected 6 out of every 10 errors systematically seeded in the outputs of the APIs under test. Additionally, AGORA revealed 11 bugs in APIs with millions of users: Amadeus, GitHub, Marvel, OMDb and YouTube. Our reports have guided developers in improving their APIs, including bug fixes and documentation updates in GitHub. Since it operates in black-box mode, AGORA can be seamlessly integrated into existing API testing tools.

1 PROBLEM AND MOTIVATION

Web Application Programming Interfaces (APIs) allow heterogeneous software systems to interact over the network [27]. Modern web APIs typically adhere to the REpresentational State Transfer (REST) architectural style, being referred to as *REST APIs* [15]. REST APIs are decomposed into multiple resources (e.g., a *payment* in the VISA API) that clients can manipulate through HTTP interactions. REST APIs have become the de facto standard for software integration, being a key part of the business model of companies such as Amazon, Google, Netflix, or Twitter. The importance and pervasiveness of REST APIs is reflected on the number of APIs hosted by popular API repositories such as RapidAPI (40K) [5].

The critical role of REST APIs in software integration has driven the creation of numerous techniques and tools for the automated detection of faults within these APIs [20, 22]. Most techniques adopt a black-box approach, where test cases are automatically derived from the specification of the API under test, typically in the OpenAPI Specification (OAS) format [4], that describes the API functionality in terms of the operations supported, as well as their input parameters and responses. These test cases are created by setting values to the input parameters and checking the validity of the returned responses by applying different test oracles, i.e., mechanisms to determine whether a test execution reveals a fault [10]. Despite the capabilities of these tools for detecting faults in industrial APIs [8, 9, 19, 25], they are all limited by the types of failures that they can detect, namely crashes (responses with a 5XX HTTP status code) [7, 8, 23], disconformities with the API specification (e.g., missing output JSON property) [7, 23], regressions [19], and violations of API best practices (e.g., checking that the results of multiple calls to idempotent operations are identical) [9]. As an example, Listing 1 shows a response for the "getAlbumTracks" operation of the Spotify API. This operation takes as input an album id (`id` parameter), a country code (`market` parameter) and a maximum number of results to return (`limit` parameter) and returns the list of tracks of the album. The response format conforms to the API specification and therefore would be considered as a correct output by existing tools. However, the response could still contain errors that would go unnoticed by current tool support, including incorrect field length or format, and violations of numerical constraints or array properties, among others. Recent surveys [20] and tool comparisons [22, 25] have identified the generation of test oracles as one of the major challenges in the generation of test cases for REST APIs. This is the problem that motivates our work.

To tackle this problem, we present AGORA, a black-box approach for the Automated Generation of Oracles for REST APIs. AGORA relies on the detection of likely invariants (i.e., properties of the API response that should always hold). For this purpose, we extended and modified the Daikon [14] system for dynamic invariant detection in two directions. Firstly, we present a novel software tool—a Daikon *instrumenter* called Beet—that converts any OAS specification and a set of API requests and responses into a format processable by Daikon. This makes our approach seamlessly integrable into existing API testing tools supporting OAS. Secondly, we further enhanced the capabilities of Daikon by customizing and expanding its default set of invariants, based on an analysis of a set of 702 operations from 40 realistic APIs that was systematically collected by the author for a previous publication [6]. Currently, AGORA supports the detection of 105 distinct types of invariants in REST APIs.

Evaluation results using 11 operations from 7 industrial APIs showed that a diverse set of just 50 API requests (and their corresponding responses) is sufficient for AGORA to learn hundreds of accurate invariants (test oracles), achieving a precision of 73.2% (81.2% when learning from 10K API requests). These results surpass those obtained using the default set of invariants in Daikon, with a precision under 52%. We also evaluated the effectiveness of the generated test oracles in detecting failures by automatically seeding 110K errors in the outputs of the API operations under test. The test oracles generated by AGORA, learned from only 50 API requests,

```json
{
    "total": 14,
    "href": "https://api.spotify.com/albums/4Em5W5HgYEvhpc/tracks
        ?limit=1&market=ES",
    "items": [
        {
            "artists": [
                {
                    "id": "2CvCyf1gEVhI0mX6aFXmVI",
                    "name": "Paul Simon"
                },
                {
                    "id": "70cRZdQywnSFp9pnc2WTCE",
                    "name": "Arthur Garfunkel"
                }
            ],
            "available_markets": [ "ES", "US", "JP" ],
            "id": "0gFvkiT2afIcJwNxXQ7W51",
            "name": "Mrs. Robinson",
            "explicit": false,
            "linked_from": {
                "id": "98cZPdKywnMGp8fnw2XTYU",
                "uri": "https://spotify.com/artist/98cZPdKywnMGp8fnw2XTYU"
            }
        }
    ]
}
```

Listing 1: Spotify API response in JSON format.

were able to detect 57.2% of the incorrect outputs, supporting the cost-effectiveness of our approach.

During our evaluation, AGORA generated several invariants that indicated issues within the target APIs. Overall, AGORA resulted in the detection of 11 faults (4 confirmed, 2 fixed) in 7 operations of 5 industrial APIs, all of which would have passed unnoticed by current test case generators.

2 BACKGROUND AND RELATED WORK

This section introduces the key concepts related to automated testing of REST APIs, test oracle generation, and Daikon.

2.1 Automated testing of REST APIs

The majority of approaches for automated testing of REST APIs adopt a black-box approach (i.e., they do not access the source code) [6, 8, 9, 18, 19, 24]. Given an OAS document, these techniques automatically generate pseudo-random test cases (sequences of HTTP requests) and test oracles (assertions on the HTTP responses).

In terms of fault detection, generated test oracles are primarily limited to detecting API crashes (e.g., 500 status codes) and violations of the API specification [7, 23]. Other test oracles focus on detecting regressions [19] or adherence to best design practices [9]. However, all these approaches have limitations in detecting issues that go beyond mere syntax. For example, existing approaches would ignore domain-specific assertions in Listing 1, such as checking that the `linked_from.uri` response field should be a valid URL that contains the value of the `linked_from.id` field, or that the size of the `items` response field should be lower or equal than the value of the `total` response field, among others. Generating such test oracles is the goal of AGORA.

2.2 Test oracle generation

Automated test case generation techniques can be classified based on their inputs, and their application domains. Regarding their inputs, test oracles have been derived from source code [12], formal specifications [16], semi-structured documentation [11], previous program executions [21, 26], or a combination of them.

A common approach for the generation of test oracles is through the detection of likely invariants. An *invariant* is a property that is always satisfied at one or more points of the execution of a program [13]. For example, given a Java function that receives an array and returns the same array with an additional element, an invariant could specify that the returned array always has a greater size than the array provided as input, i.e., `size(return.array[])>size(input.array[])`. Invariants can serve as test oracles to determine the correctness of a program output. Invariants can be detected either statically (analyzing the source code, without executing it) [17] or dynamically (analyzing the behavior of a program through multiple executions) [13, 14]. Dynamically detected invariants are referred to as *likely* invariants, until they are confirmed by a domain expert.

2.3 Daikon

Daikon [14] is an active open-source tool that detects likely invariants in programs by monitoring test executions. This monitoring process involves observing the program state at designated *program points*, initially considering all possible invariants as valid. Those invariants that are not violated by any execution are reported as likely invariants. Daikon operates by analyzing an instrumented version of a software execution, generated by an *instrumenter* or front-end. This instrumentation produces a *declaration* file (describes the structure of program points in terms of input and output variables) and a *data trace* file (contains the values assigned to the variables in each execution). Instrumenters are available for various programming languages and data formats, including Java, Perl, C++, and CSV [2].

3 UNIQUENESS OF THE APPROACH

Figure 1 shows an overview of AGORA, our approach for the automated generation of test oracles for REST APIs. At the core of the approach is Beet, a novel Daikon instrumenter. Beet receives three inputs: 1) the OAS specification of the API under test, 2) a set of API requests, and 3) the corresponding API responses. As a result, Beet returns an instrumentation of the API requests consisting on a declaration file—describing the format of the API operations inputs and outputs—and a data trace file—specifying the values assigned to each input parameter and response field in each API call. This instrumentation is then processed by our customized version of Daikon, resulting in a set of likely invariants that, once confirmed by developers, can be used as test oracles.

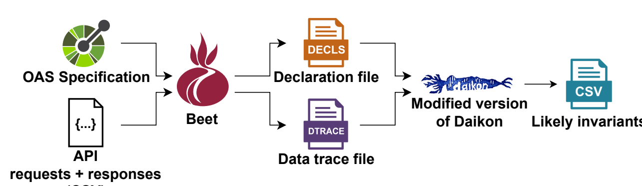

Figure 1: Workflow of AGORA.

Beet has been implemented in Java and is open-source. We refer the reader to GitHub for a exhaustive description of the instrumentation process with examples [1].

3.1 Invariant definition

This section details the changes performed on Daikon for detection of likely invariants in REST APIs. In order to identify classes of invariants that could be used as test oracles, we used a benchmark of 40 APIs (702 operations) systematically collected by the author from the RapidAPI repository [5] for a previous publication [6]. Specifically, we studied the input and output format of each operation to identify common types of invariants in REST APIs.

We implemented 22 new types of invariants, suppressed 36 default Daikon invariants, and activated 9 invariants disabled by default in Daikon. The new API-specific invariants aim to detect specific common string patterns (e.g., URLs, dates, or length constraints). Suppressed invariants would most likely provide irrelevant or misleading information in our context and thus they were disabled, such as comparing the scalar value of strings. Finally, we activated 9 invariants related to detecting subsets and supersets when comparing array variables (e.g., `x[] is a subsequence of y[]`) and detecting substrings relations between string variables (e.g., `input.id is a substring of return.href`). Our customized version of Daikon supports a total of 105 distinct types of invariants for REST APIs, classified into five categories:

- *Arithmetic comparisons (48 invariants).* Specify numerical bounds (e.g., `size(return.artists[]) >= 1`) and relations between numerical fields (e.g., `input.limit >= size(return.items[])`).
- *Array properties (23 invariants).* Represent comparisons between arrays, such as subsets, supersets, or fields that are always member of an array (e.g., `return.hotel.hotelId in input.hotelIds[]`).
- *Specific formats (22 invariants).* Specify restrictions regarding the expected format (e.g., `return.href is Url`) or length (e.g., `LENGTH(return.id)==22`) of string fields.
- *Specific values (9 invariants).* Restrict the possible values of fields (e.g., `return.visibility one of {"public", "private"}`).
- *String comparisons (3 invariants).* Specify relations between string fields, such as equality (e.g., `input.name == return.name`) or substrings (e.g., `input.id is a substring of return.href`).

We refer the reader to the AGORA GitHub repository [1] for a more detailed description of each type of invariant, including examples. The set of invariants is not exhaustive and more types of invariants could be considered in the future.

Listing 2 shows some of the likely invariants returned by AGORA for the "getAlbumTracks" operation of the Spotify API after learning invariants from several responses such as the one depicted in Listing 1.

```
=== getAlbumTracks&200():::ENTER
LENGTH(input.id)==14
input.limit >= 1
LENGTH(input.market)==2
=== getAlbumTracks&200():::EXIT
return.href is Url
input.limit >= size(return.items[])
return.total >= size(return.items[])
return.total >= 1
input.market is a substring of return.href
input.id is a substring of return.href
=== getAlbumTracks&200&items():::ENTER
...
=== getAlbumTracks&200&items():::EXIT
size(return.artists[]) >= 1
All the elements of return.available_markets[] have LENGTH=2
input.market in return.available_markets[]
LENGTH(return.id)==22
LENGTH(return.linked_from.id)==22
return.linked_from.uri is Url
LENGTH(return.linked_from.uri)==54
return.linked_from.id is a substring of return.linked_from.uri
=== getAlbumTracks&200&items&artists():::ENTER
...
=== getAlbumTracks&200&items&artists():::EXIT
LENGTH(return.id)==22
```

Listing 2: Detected invariants.

4 RESULTS AND CONTRIBUTIONS

4.1 Evaluation Results

For our experiments, we resorted to a set of 11 operations from 7 industrial APIs. For each operation, we automatically generated and executed API calls using the RESTest [23] framework until obtaining 10K valid API calls per operation (110K calls in total). According to REST best practices [27], we consider an API response as valid if it is labeled with a 2XX HTTP status code. We performed the following experiments to assess the effectiveness of AGORA for generating of test oracles and detecting failures:

4.1.1 *Experiment 1: Test oracle generation.*

For each API operation, we randomly divided the set of automatically-generated requests (10K) into subsets of 50, 100, 500, 1K and 10K requests. Then, we ran AGORA—Beet instrumentation plus customized Daikon execution—using each subset as input and computed precision by manually classifying the inferred invariants as true or false positives. True positive invariants describe properties of the output that should always hold and therefore are valid test oracles. A false positive reflects a pattern that has been observed in all the API requests and responses provided as input but does not represent the expected behavior of the API. For example, one of the likely invariants inferred in Spotify states that the duration in milliseconds of a song should always be greater than the number of artists in the song: (`return.duration_ms>size(return.artists[])`). While this may be true in most cases, it is clearly not the intended behavior of the API, and therefore it is considered a false positive.

We could not find any comparable approach to be used as a baseline. Therefore, we compared the effectiveness of AGORA against the default version of Daikon in identifying likely invariants for REST APIs. In both cases, Beet was used to transform API specifications, requests, and responses into inputs for Daikon. With this experiment, we aim to answer the following research questions:

RQ1: Effectiveness of AGORA to generate test oracles. Table 1 shows the results for each API operation, set of API requests (50, 100, 500, 1K, 10K) and approach (AGORA vs default Daikon). The columns labeled with "I" present the number of likely invariants detected, whereas the columns labeled with "P" present the total precision, that is, the percentage of true positives.

When learning from the whole dataset (10K API requests), AGORA obtained a total precision of 81.2% (770 out of 948 invariants are valid oracles), whereas the original configuration of Daikon achieved 51.4% (363 out of 706). AGORA outperformed the default configuration of Daikon in all the API operations. Precision ranged between 50% in the Yelp API and 100% in the "createPlaylist" operation of the Spotify API. The number of invariants reported per operation oscillated between 7 in the "bySearch" operation of the OMDb API and 198 in the "createOrganizationRepository" of the GitHub API. We observed a correlation between the number of response fields and the number of reported invariants. This was confirmed by a correlation study with a Spearman coefficient of 0.9, indicating that the number of invariants tends to increase with the response size.

The largest portion of false positives (75.8%) were found in the arithmetic comparison category, followed by specific values (18.5%), specific formats (2.8%) and string comparisons (2.8%), with no false positives of the array properties category. It is noteworthy that the precision of AGORA increases to 93.6% when suppressing the invariants of the arithmetic comparison category.

False positives in the arithmetic comparison category typically occur when comparing numerical fields with values in different orders of magnitude (e.g., duration of a Spotify song in milliseconds and its number of artists). In many cases, it may be difficult to find a counterexample that refutes these invariants.

RQ2: Impact of the size of the input dataset to the precision of AGORA. The test suites of 50 requests seem to offer the best trade-off between effectiveness and cost of generating API requests. The total precision of AGORA improved from 73.2% with 50 API requests, to 81.2% with the complete dataset. This means a drop in precision of just 8.1% when using the smallest dataset against the complete one. The precision of AGORA for the 50 API requests sets increases to 87.6% when excluding arithmetic comparisons.

4.1.2 Experiment 2: Failure detection. To evaluate the effectiveness of the generated test oracles in detecting failures (i.e., erroneous outputs) in the APIs under test, we systematically seeded *errors* in API responses using JSONMutator [3], an open-source mutation tool that applies different mutation operators on JSON data, e.g., removing an array item. This approach differs from traditional mutation testing, where *faults* are seeded in the source code of the program under test. The motivation behind our strategy is to assess the failure detection capabilities of the generated test oracles on large-scale industrial APIs, for which source code is not available.

For each API operation, we selected the test oracles derived from the set of 50 test cases. Test oracles were transformed into executable assertions in Java, 724 in total (Table 2). Then, for each API operation, we randomly selected 1K API responses from the set of 10K test cases generated by RESTest meeting the following constraints: (1) they were not part of the 50-requests set used for detecting the invariants, (2) they contained at least one result item (we cannot apply mutation operators on empty arrays), and (3) they revealed no failures (c.f. Section 4.2).

We used JSONMutator to introduce a single error on each API response simulating a failure. Then, we ran the assertions and marked the failure as detected if at least one of the test assertions (i.e., test oracles) was violated. We repeated this process 10 times per operation to minimize the effect of randomness computing the average percentage of failures detected. In total, the results are based on 110K seeded errors: 11 operations x 1,000 API responses x 10 repetitions.

For our experiments, we configured JSONMutator to apply mutation operators that resulted in syntactically valid mutants, i.e., conform to the API specification. Syntactically invalid mutants (e.g., adding a new property to a JSON object) can be detected by existing approaches and therefore are out of the scope of AGORA. Specifically, we enabled the mutation operators that consist of changing Boolean, double, long and string values (e.g., adding or removing characters) and altering array values (e.g., removing and disordering elements), using a total of 12 mutation operators. All the mutations resulted in a distinguishable change in the API response and therefore there were no equivalent mutants. We disabled the mutation operators that produced mutants non-conformant with the OAS specification. Also, we disabled operators that converted response fields into null values, since null values are easily detected as a violations of the `nullable` property of OAS. With this experiment, we aim to answer the following research question:

RQ3: Effectiveness of the generated test oracles for detecting failures. Table 2 shows the number of test assertions (i.e., test oracles) and the percentage of detected failures for each API operation. Overall, test oracles generated by AGORA identified 57.2% of the failures. This percentage ranged between 20.6% in the "bySearch" operation of the OMDb API and 92.3% in the "createOrganizationRepository" of the GitHub API.

4.2 Detected Faults

The invariants detected by AGORA allowed us to detect bugs in some of the APIs under test, showing the potential of the approach as a testing technique on its own. Some of the invariants revealed inconsistent behavior, e.g., hotel rooms with *zero* beds. We also found cases where a confirmed invariant (i.e., test oracle) was discarded when increasing the size of the input dataset, meaning that a counterexample (i.e., failure) had been detected. Overall, AGORA detected 11 domain specific bugs in 7 operations from 5 APIs with millions of users worldwide. Next, we detail the detected bugs.

Amadeus Hotel. One of the detected invariants in the Amadeus Hotel API led to the identification of 55 hotel offers in which the offered room had zero beds (`return.room.typeEstimated.beds>=0`). This bug has been confirmed and fixed by Amadeus developers.

GitHub. In the "createOrganizationRepository" operation, the violation of the confirmed invariant `input.license_template==return.license.key` revealed 15 test cases in which the repository is created with an incorrect license. This bug has been confirmed by the API providers. Also, contrary to what is stated in the API specification and the documentation, AGORA detected that the field `template_directory` was never included in the responses of the "getOrganizationRepositories" operation (`return.template_repository==null`). Developers confirmed the issue and updated the documentation of GitHub accordingly.

Marvel. In the "getComicById" operation, AGORA detected +3.1K comics with 0 pages (`return.pageCount>=0`), invalid date formats, comics with an invalid Diamond code (violations of the invariant `LENGTH(return.diamondCode)==9`), and invalid values for the EAN code (`LENGTH(return.ean)==20`). For example, we found a case where the EAN code had the value of the Diamond code. These reports have not been confirmed yet.

OMDb. The `type` parameter of the OMDb API operations is used to filter the obtained results to one media type: "movie", "series" or "episode" (according to the documentation). However, one of the invariants (`return.Type one of {"game","movie","series"}`) revealed a new value for this parameter that was not specified in the documentation: "game". Moreover, we detected that none of

Table 1: Test oracle generation. I="Number of likely invariants", P="Precision (% valid test oracles)"

API - Operation	50 API calls				100 API calls				500 API calls				1K API calls				10K API calls			
	Daikon		AGORA		Daikon		AGORA		Daikon		AGORA		Daikon		AGORA		Daikon		AGORA	
	I	P (%)	I	P (%)	I	P (%)	I	P (%)	I	P (%)	I	P (%)	I	P (%)	I	P (%)	I	P (%)	I	P (%)
AmadeusHotel-getMultiHotelOffers	109	21.1	117	52.1	136	16.9	114	56.1	116	22.4	108	64.8	107	24.3	107	66.4	99	26.3	106	67.9
GitHub-createOrganizationRepository	82	95.1	198	98	82	95.1	198	98	80	96.2	198	98.5	80	96.2	198	98.5	80	96.2	198	98.5
GitHub-getOrganizationRepositories	45	40	150	84.7	40	45	147	88.4	39	46.2	149	87.9	39	46.2	150	88	38	47.4	148	89.2
Marvel-getComicById	178	29.8	115	47.8	194	28.9	127	46.5	178	33.7	119	52.1	167	35.9	106	58.5	140	45.7	96	65.6
OMDB-byIdOrTitle	7	57.1	16	93.8	7	57.1	16	93.8	7	57.1	16	93.8	8	50	17	88.2	7	57.1	16	93.8
OMDB-bySearch	4	100	5	100	7	57.1	7	71.4	5	80	6	83.3	5	80	6	83.3	6	83.3	7	85.7
Spotify-createPlaylist	22	100	41	100	22	100	41	100	22	100	41	100	22	100	41	100	22	100	41	100
Spotify-getAlbumTracks	46	45.7	68	85.3	45	46.7	67	86.6	42	50	66	87.9	42	50	66	87.9	41	53.7	66	89.4
Spotify-getArtistAlbums	53	43.4	55	81.8	49	49	52	88.5	35	68.6	50	92	32	75	50	92	31	83.9	52	92.3
Yelp-getBusinesses	60	28.3	30	40	55	30.9	33	36.4	46	37	25	48	45	37.8	23	52.2	41	39	22	50
YouTube-listVideos	228	31.6	194	57.2	227	32.2	199	56.3	218	35.8	191	62.3	225	36	200	61.5	201	41.3	196	65.3
TOTAL	**834**	**40.2**	**989**	**73.2**	**864**	**39.4**	**1001**	**73.5**	**788**	**44.5**	**969**	**77.8**	**772**	**45.9**	**964**	**78.8**	**706**	**51.4**	**948**	**81.2**

Table 2: Failure Detection Ratio per API operation.

API - Operation	Assertions (test oracles)	FDR (%)
AmadeusHotel-getMultiHotelOffers	61	60
GitHub-createOrganizationRepository	194	92.3
GitHub-getOrganizationRepositories	127	62.9
Marvel-getComicById	55	37.7
OMDB-byIdOrTitle	15	36.5
OMDB-bySearch	5	20.6
Spotify-createPlaylist	41	84.8
Spotify-getAlbumTracks	58	70.6
Spotify-getArtistAlbums	45	76.5
Yelp-getBusinesses	12	23.5
YouTube-listVideos	111	64.2
TOTAL	**724**	**57.2**

the operations support filtering by "episode". These reports have not been confirmed yet.

YouTube. When performing a search using the `regionCode` input parameter, the returned videos must be available in the provided region. However, a violation of the confirmed invariant `input.regionCode in return.contentDetails.regionRestriction.allowed[]`, led us to detect 81 cases in which the API returned videos that were not available in the provided region. This error has been confirmed by YouTube developers.

4.3 Contributions

This paper presents the following original research and engineering contributions:

- AGORA, a black-box approach for the automated generation of test oracles for REST APIs based on the analysis of the API specification and previous requests and responses.
- Beet [1], a publicly available Daikon instrumenter for REST APIs readily integrable into existing test case generation tools for REST supporting OAS.
- A customized version of Daikon supporting the detection of 105 distinct types of invariants in REST APIs.
- An empirical evaluation of AGORA in terms of precision and failure detection in 11 operations from 7 industrial APIs, including reports of 11 real-world bugs.

REFERENCES

[1] 2023. Beet repository. https://github.com/isa-group/Beet
[2] 2023. DAIKON instrumenters. https://plse.cs.washington.edu/daikon/download/doc/daikon.html#Front-ends-_0028instrumentation_0029.
[3] 2023. JSONMutator. https://github.com/isa-group/JSONmutator.
[4] 2023. OpenAPI Specification. https://www.openapis.org.
[5] 2023. RapidAPI API directory. https://rapidapi.com/marketplace.
[6] Juan C. Alonso, Alberto Martin-Lopez, Sergio Segura, Jose Maria Garcia, and Antonio Ruiz-Cortes. 2022. ARTE: Automated Generation of Realistic Test Inputs for Web APIs. *IEEE TSE* (2022).
[7] Andrea Arcuri. 2019. RESTful API Automated Test Case Generation with EvoMaster. *ACM TOSEM* (2019).
[8] V. Atlidakis, P. Godefroid, and M. Polishchuk. 2019. RESTler: Stateful REST API Fuzzing. In *2019 IEEE/ACM 41st ICSE.*
[9] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2020. Checking Security Properties of Cloud Services REST APIs. In *2020 IEEE ICST.*
[10] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE TSE* (2015).
[11] Arianna Blasi, Alessandra Gorla, Michael D. Ernst, Mauro Pezzè, and Antonio Carzaniga. 2021. MeMo: Automatically identifying metamorphic relations in Javadoc comments for test automation. *Journal of Systems and Software* (2021).
[12] Elizabeth Dinella, Gabriel Ryan, Todd Mytkowicz, and Shuvendu K. Lahiri. 2022. TOGA: A Neural Method for Test Oracle Generation. In *ICSE 2022.* ACM.
[13] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. 2001. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering* 27, 2 (Feb. 2001), 99–123.
[14] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming* 69, 1 (2007), 35–45. Special issue on Experimental Software and Toolkits.
[15] Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures.* Ph. D. Dissertation. University of California, Irvine.
[16] Gregory Gay, Sanjai Rayadurgam, and Mats P.E. Heimdahl. 2014. Improving the Accuracy of Oracle Verdicts through Automated Model Steering. In *ASE 2014.* ACM.
[17] C. Giuffrida, L. Cavallaro, and A. S. Tanenbaum. 2013. Practical automated vulnerability monitoring using program state invariants. In *2013 IEEE/IFIP DSN.* IEEE Computer Society.
[18] Patrice Godefroid, Bo-Yuan Huang, and Marina Polishchuk. 2020. Intelligent REST API Data Fuzzing. In *ACM ESEC/FSE 2020.*
[19] Patrice Godefroid, Daniel Lehmann, and Marina Polishchuk. 2020. Differential Regression Testing for REST APIs. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020).* Association for Computing Machinery, New York, NY, USA, 312–323.
[20] Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2022. Testing RESTful APIs: A Survey.
[21] Ali Reza Ibrahimzada, Yigit Varli, Dilara Tekinoglu, and Reyhaneh Jabbarvand. 2022. Perfect is the Enemy of Test Oracle. In *ESEC/FSE 2022.* ACM.
[22] Myeongsoo Kim, Qi Xin, Saurabh Sinha, and Alessandro Orso. 2022. Automated Test Generation for REST APIs: No Time to Rest Yet. In *ISSTA 2022.* ACM.
[23] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2020. RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs. In *ICSOC 2020.*
[24] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2021. RESTest: Automated Black-Box Testing of RESTful Web APIs. In *ISSTA 2021.*
[25] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. 2022. Online Testing of RESTful APIs: Promises and Challenges. In *ESEC/FSE 2022.* ACM.
[26] Facundo Molina, Marcelo d'Amorim, and Nazareno Aguirre. 2022. Fuzzing Class Specifications. In *ICSE 2022.* ACM.
[27] Leonard Richardson, Mike Amundsen, and Sam Ruby. 2013. *RESTful Web APIs.* O'Reilly Media, Inc.