TRABAJO FIN DE MÁSTER

Generación Automática de Oráculos de Prueba para APIs REST

Realizado por
Juan Carlos Alonso Valenzuela

Para la obtención del título de
Máster en Ingeniería del Software: Cloud, Datos y Gestión TI

Dirigido por
Sergio Segura Rueda

Convocatoria de Julio, curso 2021/22

Agradecimientos

Antes de presentar el trabajo que he realizado, me gustaría utilizar esta página para agradecer personalmente a todas las personas que lo han hecho posible.

A mi tutor, Sergio Segura Rueda, por iniciarme en el mundo de la investigación y la docencia que tanto me apasiona, por sus consejos y, en general, por ser el mejor tutor que se podría desear. Estoy deseando que continuemos las líneas de investigación derivadas de este proyecto y de mis trabajos previos para la realización de mi tesis doctoral.

A todos los compañeros de mi grupo de investigación, por hacerme sentir como uno más desde el primer día, especialmente a aquellos que me han acompañado y aconsejado durante mi primer año como profesor, y a los demás estudiantes de doctorado de la línea de pruebas automáticas, Alberto Martín y Giuliano Mirabella.

A mis padres y a mi abuela, por su apoyo incondicional durante toda mi vida y por hacer posible mi carrera académica.

A mi grupo de amigos, Ramón, Roberto, Samuel, Emilee, Emily, Julio y Peter, por darme ánimos en los momentos más duros de este año tan peculiar y por su paciencia por mi poca disponibilidad en estos últimos meses.

A mis compañeros del grado y del máster, con los que me alegro de poder mantener todavía una relación de amistad: Alex, Vanessa, Gabie, David Romero, Germán, José Sandoval, Roberto, Chus, David Gil, Pablo García y Dani.

A todos vosotros, muchas gracias.

Resumen

La generación automática de casos de prueba para APIs RESTful es un área de investigación en auge, dado el rol que desempeñan en la integración software. La mayoría de las propuestas en este ámbito siguen un enfoque de caja negra, en el que los casos de prueba se generan automáticamente a partir de la especificación de la API. A pesar de los prometedores resultados de estas propuestas, todas se encuentran limitadas por los tipos de errores que pueden detectar: fallos de servidor no controlados (etiquetados con un código 500) y disconformidades con la especificación de la API. La falta de técnicas para la detección de errores específicos del dominio de cada API supone una importante limitación para el grado de automatización obtenido por estas herramientas.

En este trabajo, se propone un enfoque basado en la detección de invariantes (i.e., propiedades que siempre se cumplen en uno o más puntos de la ejecución de un programa) para automatizar el proceso de generación de oráculos de prueba, que pueden ser utilizados para la creación de assertions. En concreto, la propuesta recibe como entrada la especificación de la API y un conjunto de pruebas generadas automáticamente (para las que se conocen únicamente los valores de las entradas y la salida devuelta) y devuelve un conjunto de invariantes que pueden ser utilizados como oráculos.

Los resultados obtenidos en una evaluación realizada sobre un conjunto de 8 operaciones de 6 APIs comerciales muestran la capacidad de la propuesta para generar cientos de oráculos válidos, llegando a obtener una precisión del 100 % (y una precisión total del 66.5 %) y detectando un total de 6 bugs replicables en 4 operaciones pertenecientes a 3 sistemas con millones de usuarios (Amadeus Hotel, GitHub y OMDb).

Palabras clave: APIs RESTful, Problema del oráculo, Detección de invariantes

Abstract

Automated test case generation for RESTful APIs is a thriving research topic, given the key role they play in software integration. Most proposals in this area follow a black-box approach, where test cases are automatically generated from an API specification. Despite the promising results of these proposals, they are all limited by the types of errors they can detect: uncontrolled server failures (labelled with a 500 status code) and disconformities with the API specification. The lack of domain-specific error detection techniques for each API is a major limitation to the degree of automation obtained by these tools.

In this paper, we propose an approach based on the detection of invariants (i.e., properties that are always satisfied at one or more points in the execution of a program) to automate the process of generating test oracles, which can be used for the creation of assertions. Specifically, the proposal receives as input the API specification and a set of automatically generated tests (for which only the values of the inputs and the returned output are known) and returns a set of invariants that can be used as oracles.

The results obtained in an evaluation performed on a set of 8 operations from 6 commercial APIs show the capability of the proposal to generate hundreds of valid oracles, obtaining an accuracy of up to 100 % (and a total accuracy of 66.5 %) and detecting a total of 6 reproducible bugs in 4 operations belonging to 3 systems with millions of users (Amadeus Hotel, GitHub and OMDb).

Keywords: RESTful APIs, Oracle problem, Invariant detection

Índice general

Índice de figuras

Índice de extractos de código

1. Introducción

En la actualidad, es habitual que distintas aplicaciones software (en muchas ocasiones pertenecientes a distintos proveedores) se comuniquen entre sí a través de Internet. Esta comunicación tiene lugar generalmente mediante el uso de las conocidas como APIs web (Web Application Programming Interfaces).

La mayoría de las APIs web actuales siguen el estilo arquitectónico REST (REpresentational State Transfer) [45], siendo conocidas como APIs RESTful. Las APIs RESTful se han convertido en el estándar de-facto para la integración web, compañías como Google, Amazon o Microsoft utilizan APIs RESTful para proveer la mayoría de sus servicios. Es muy común que estas compañías, así como desarrolladores independientes, ofrezcan acceso público a sus APIs para permitir su integración en aplicaciones de terceros. En la práctica, esto permite incluir en nuestras aplicaciones funcionalidades tan diversas como comprobar los resultados de un partido de fútbol (BeSoccer API [6]), escribir un tweet (Twitter API [29]), reservar una habitación de hotel (Amadeus API [4]), traducir un texto (DeepL API [9]), o encontrar una ruta entre dos ubicaciones (Openroute API [18]). El amplio catálogo de APIs de distintas categorías disponibles en repositorios como Programmable web [19] y RapidAPI [20] (que actualmente alojan 24K y 30K APIs respectivamente), reflejan la importancia del papel de las APIs RESTful en la integración web.

Dada la importancia del rol que desempeñan las APIs RESTful en la integración web, resulta vital probarlas de forma exhaustiva, ya que un fallo en una sola API puede tener consecuencias en todos los sistemas que la integran. Afortunadamente, todas las APIs RESTful siguen unas mismas directrices de diseño claramente definidas, por lo que en muchos casos una misma técnica para la generación de pruebas pueda ser aplicada a cualquier API, independientemente de la tecnología usada para su implementación.

Por estas razones, en años recientes han proliferado una gran cantidad de técnicas orientadas al testing automático de APIs RESTful. Estas técnicas generan casos de prueba automáticamente a partir de una especificación OAS (OpenAPI Specification) [17], que describe la funcionalidad de una API en términos de las operaciones que soporta, los parámetros que recibe y los formatos de respuesta. Sin embargo, tal y como se señala en un estudio comparativo de herramientas centradas en la generación automática de pruebas para APIs RESTful publicado recientemente [51], una de las principales limitaciones compartida por todas las herramientas que conforman el estado del arte reside en el tipo de errores que pueden detectar: fallos inesperados (i.e., fallos del servidor etiquetados con un código 500), y errores de conformidad con la especificación OAS (por ejemplo, que el valor de un campo de la respuesta que debería ser de tipo numérico sea un string). No obstante, pueden existir toda clase de inconsistencias en respuestas aparentemente válidas que no serían detectadas por ninguna de estas propuestas. Por ejemplo, si un campo de la respuesta que indique la edad de un usuario tiene un valor negativo, si un string que representa la URL de un repositorio no es una URL válida o si al realizar una búsqueda de ubicaciones filtrando por países las ubicaciones devueltas no pertenecen al país indicado como parámetro.

1.1. Problema

Determinar si la respuesta devuelta por una API es correcta o no constituye un desafío de gran complejidad, ya que en la mayoría de las ocasiones no es posible conocer su salida esperada, esto es lo que se conoce como el *problema del oráculo* [39]. Tomemos como ejemplo la operación "Get Album tracks" de la API de Spotify. Esta operación recibe como parámetro el id de un álbum de Spotify (parámetro `id`) y el número máximo de resultados a devolver (parámetro `limit`).

Tras procesar estos parámetros, la API devuelve una respuesta similar a la mostrada en el Extracto de código 1.1. A partir de la especificación OAS y sin poseer ningún conocimiento del dominio, no podemos detectar automáticamente si la respuesta es correcta más allá de comprobar si cada campo de la respuesta es del tipo indicado (por ejemplo, `href` debe ser de tipo string). Sin embargo, observando esta respuesta, podemos pensar en algunos oráculos que deberían cumplirse para todas las respuestas de la API:

- Los valores de los campos `limit` (número máximo de resultados que pueden devolverse) y `total` (número total de canciones del album) deben ser mayores o iguales que el número de resultados devueltos por la API (i.e., el tamaño del array `items`). Asimismo, tampoco pueden ser negativos.
- El valor del campo `limit` debe ser igual que el valor seleccionado para el parámetro `limit` al hacer la llamada a la API.
- Ciertos parámetros de tipo string deben tener un formato específico. Este es el caso, por ejemplo, de las URLs (campos `href`, y `preview_url`), los ids de bases de datos (como los campos `id` de cada objeto de tipo "track" o "artist") o los códigos de países (todos los elementos del array de strings `available_markets` deberían tener longitud 2).
- El campo `type` siempre debe tener un valor específico ("artist" para el objeto de tipo artista y "track" para el objeto de tipo canción).
- Toda canción debería tener al menos un artista. En otras palabras, el número de elementos de la propiedad `artists` debería ser siempre mayor o igual que 1.

```
{
    "limit": 20,
    "total": 18,
    "href": "https://api.spotify.com/v1/albums/4aawyAB9vmqN3uQ7FjRGTy/tracks?limit=20",
    "items": [
        {
            "artists": [
                {
                    "href": "https://api.spotify.com/v1/artists/0TnOYISbd1XYRBk9myaseg",
                    "id": "0TnOYISbd1XYRBk9myaseg",
                    "name": "Pitbull",
                    "type": "artist"
                }
            ],
            "available_markets": [ "ES", "IT", "US" ],
            "id": "6OmhkSOpvYBokMKQxpIGx2",
            "name": "Global Warming (feat. Sensato)",
            "track_number": 1,
            "type": "track",
            "preview_url": "https://p.scdn.co/mp3-preview/4df38b27b145e6e2d180a0790e991af3eef99d86",
            ...
        },
        ...
    ]
}
```

Extracto de código 1.1: Ejemplo de respuesta de Spotify.

Aunque es posible definir oráculos que describan el comportamiento esperado de la API, como los mostrados anteriormente, estos son específicos de la API que se está probando, y su generación requiere de trabajo manual y conocimiento del dominio. Esto supone una limitación enorme para las actuales técnicas de generación de pruebas, reduciendo su efectividad de forma significativa.

1.2. Contribución

En este trabajo proponemos una técnica y una herramienta para la generación automática de oráculos de pruebas para APIs RESTful, como los mostrados en la sección anterior. Para ello, partiremos de un conjunto de llamadas válidas, modelando la generación de oráculos como un problema de detección de invariantes potenciales. Un invariante es una propiedad que se cumple en un punto o en una serie de puntos de la ejecución de un programa, como pueden ser sus entradas y sus salidas en el contexto de las pruebas de caja negra.

Para llevar a cabo la detección de estos invariantes, se ha desarrollado un *instrumenter* que permite expresar un conjunto de pruebas en un formato procesable por una versión modificada de la librería detección de invariantes Daikon [43]. La propuesta recibe como entrada una especificación OAS y un conjunto de pruebas y devuelve un conjunto de invariantes que pueden utilizarse como oráculos.

Una evaluación con 8 operaciones de 6 APIs industriales muestra la capacidad de la propuesta para la generación automática de oráculos, obteniendo una precisión de hasta el 100 % en APIs como Spotify y OMDb (y una precisión total del 66.5 %). Además, la propuesta ha detectado un total de 6 errores o inconsistencias reales y replicables en sistemas de millones de usuarios como Amadeus, GitHub y OMDb.

Un resumen de este trabajo ha sido aceptado para su presentación en una conferencia nacional: Juan C. Alonso, Sergio Segura, and Antonio Ruiz-Cortés. *Generación*

Automática de Oráculos de Prueba para APIs RESTful, *Jornadas de la Sociedad de Ingeniería de Software y Tecnologías de Desarrollo de Software (SISTEDES)*, 2022.

Para facilitar la replicación de este trabajo, se ha creado un paquete con material suplementario que contiene el código desarrollado y los datos utilizados [21].

1.3. Metodología

Para el desarrollo de este proyecto, se ha seguido la metodología de Ciencia del Diseño [55] (en inglés, *Design Science Research Methodology*, o DSRM). DSRM se centra en la creación y mejora de artefactos como pueden ser sistemas, métodos, procedimientos y herramientas, entre otros. Estos artefactos son creados con la intención de resolver un problema en concreto, y son sometidos a un proceso continuo de evaluación con el objetivo de que resuelvan mejor el problema que deben resolver. DSRM está compuesto por seis pasos. A continuación, se definen estos seis pasos y su aplicación en el contexto de este trabajo:

- *Identificación del problema y motivación:* En esta fase, es necesario definir un problema de investigación concreto y justificar el valor de la solución. El problema afrontado en este trabajo es la falta de técnicas para la validación automática de las respuestas devueltas por las APIs RESTful. Dicho problema fue identificado durante un trabajo anterior del autor [34]. Proveer una solución a este problema sería de gran interés para la industria, ya que permitiría ahorrar tiempo y costes.
- *Definición de objetivos para obtener la solución:* Este paso consiste en enunciar el objetivo perseguido para afrontar (e idealmente resolver) el problema identificado en la fase anterior. El principal objetivo de este proyecto es la creación de una herramienta que permita generar automáticamente oráculos de prueba que permitan incrementar el nivel de automatización logrado por las herramientas que constituyen actualmente el estado del arte.
- *Diseño y desarrollo:* Durante esta fase, el artefacto es creado. El artefacto desarrollado para este proyecto es un instrumenter que permite expresar una serie de casos de prueba como un problema de detección de invariantes. Asimismo, se ha modificado la librería Daikon [43] para adecuarla al contexto del problema a resolver. Este artefacto ha sido iterativamente refinado y mejorado durante todo el desarrollo del trabajo.
- *Demostración:* Esta fase tiene el objetivo de mostrar las capacidades del artefacto desarrollado para resolver el problema que se busca resolver. En el contexto de este proyecto, este paso se ha utilizado como un análisis previo para verificar si el artefacto puede resolver un problema específico antes de pasar a una evaluación más formal.
- *Evaluación:* A lo largo de esta fase se evaluará el rendimiento del artefacto para solucionar el problema planteado al inicio del desarrollo. En el caso de este trabajo, dicha evaluación ha consistido en probar el artefacto desarrollado con una serie de sistemas comerciales y evaluar su rendimiento en términos de precisión y capacidad de detección de errores.

- *Comunicación:* Esta fase es la parte final del proceso de DSRM, donde el trabajo realizado y los resultados obtenidos son compartidos con la comunidad investigadora y otras audiencias. Los resultados de la investigación llevada a cabo en este trabajo han sido aceptados para su publicación en una conferencia nacional [35].

1.4. Estructura de este documento

El resto de este documento está organizado de la siguiente forma: el capítulo 2 provee del contexto necesario para entender la propuesta mediante una descripción de las principales tecnologías software involucradas; el capítulo 3 consiste en un estudio del estado del arte; el capítulo 4 contiene los detalles de diseño e implementación de la propuesta presentada; en el capítulo 5 se explica el proceso experimental seguido para evaluar la propuesta; el capítulo 6 muestra las distintas amenazas a la validez que podrían haber afectado al desarrollo del proyecto, y como estas han sido mitigadas. Finalmente, el capítulo 7 concluye este trabajo y presenta las principales líneas de trabajo futuro a seguir.

2. Contexto

Esta sección presenta los conceptos clave necesarios para entender el contexto de la propuesta presentada.

2.1. APIs RESTful

Una API RESTful es una API web que sigue el estilo arquitectónico REST [45]. Al consumir una API RESTful, un cliente puede comunicarse con un servicio web mediante el envío de peticiones HTTP y procesando los mensajes que recibe como respuesta, generalmente en formato JSON. La Figura 2.1 muestra un esquema de la estructura de esta comunicación.

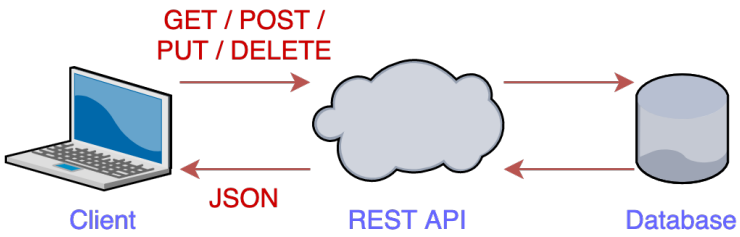

Figura 2.1: Diagrama del funcionamiento de una API RESTful

Al enviar estas peticiones, el cliente interactúa con una serie de recursos gestionados por el proveedor de la API. Un recurso representa cualquier tipo de dato sobre el que el cliente puede ejecutar una o varias operaciones CRUD (Create, Read, Update y Delete). Estas operaciones suelen estar asociadas, respectivamente, a los métodos HTTP POST, GET, PUT y DELETE. Las peticiones se envían a un endpoint de la API, que se identifica por la combinación de un path y un método HTTP. La Tabla 2.1 muestra ejemplos de distintos tipos de operaciones sobre el recurso *Playlist* de la API de Spotify.

Método HTTP	**Path**	**Acción**
POST	`/users/{user_id}/playlists`	Crear una nueva playlist
GET	`/me/playlists`	Devolver las playlists del usuario
PUT	`/playlists/{playlist_id}`	Editar playlist
DELETE	`/playlists/{playlist_id}/tracks`	Elimina los items de una playlist

Tabla 2.1: Ejemplo de CRUD

Al realizar una petición, además del método HTTP y el path, el cliente también puede enviar parámetros en el body, header, query o path de la petición. Tras recibir y procesar la petición, el servicio web devuelve el cuerpo de la respuesta (generalmente en formato JSON), etiquetado con un código HTTP de estado de tres dígitos que indica si la petición ha tenido éxito (y, en caso negativo, la razón por la que no ha tenido éxito).

Los códigos de estado pueden clasificarse en cinco grupos en función de su primer dígito [13]:

- **Códigos 1XX – Respuesta informativa:** Indican que la respuesta ha sido recibida y entendida por el servidor. Suele enviarse de forma provisional mientras el procesado de la petición continúa, indicando al cliente que espere a la respuesta final.
- **Códigos 2XX – Respuesta exitosa:** Indican que la petición ha sido correctamente recibida y procesada por el servidor.
- **Códigos 3XX – Redirección:** Indican que el cliente debe realizar acciones adicionales para completar la petición.
- **Códigos 4XX – Error de cliente:** Indican que se ha producido un error, causado por el cliente, al intentar procesar la petición enviada. Estos aparecen, entre otros casos, cuando el usuario inserta datos inválidos (código 400), cuando no está autorizado para realizar una acción (código 401) o al intentar acceder a un recurso que no existe (código 404).
- **Códigos 5XX – Error del servidor:** Indican que la respuesta no ha podido ser procesada correctamente a causa de un error del servidor.

2.1.1. Especificación OAS

La especificación OAS (OpenAPI Specification) [17] define una interfaz estándar que permite describir, en un formato legible tanto por humanos como por máquinas, las distintas operaciones y servicios ofrecidos por una API RESTful sin necesidad de acceder a su código fuente e independientemente de la tecnología usada para su implementación.

OAS no sólo proporciona una forma homogénea de describir APIs, sino que también permite automatizar diversas tareas. Por ejemplo, dada la descripción de una API con OAS, es posible generar interfaces gráficas con documentación interactiva o generar pruebas de forma automática. Adicionalmente, es posible generar automáticamente una especificación OAS a partir del código fuente de la API debidamente anotado.

```yaml
paths:
  '/albums/{id}/tracks':
    get:
      operationId: 'getAlbumTracks'
      description: 'Get Tracks of an Album'
      parameters:
        - name: id
          description: |
            'The Spotify ID for the album'
          in: path
          required: true
          type: string
        - name: market
          description: |
            'An ISO 3166-1 alpha-2 country code'
          in: query
          required: false
          type: string
        - name: limit
          description: |
            'The maximum number of items to return'
          in: query
          default: 20
          maximum: 50
          minimum: 0
          required: false
          type: integer
        - name: offset
          description: |
            'The index of the first item to return'
          in: query
          default: 0
          required: false
          type: integer

      responses:
        '200':
          description: 'OK'
          schema:
            type: object
            properties:
              href:
                description: |
                    'A link to the full result.'
                type: string
              total: ...
              offset: ...
              limit: ...
              next: ...
              previous: ...
              items:
                description: 'The requested data.'
                type: array
                items:
                  type: object
                  properties:
                    artists:
                      type: array
                      items:
                        type: object
                        properties:
                          type:
                            description: |
                                'Object type: artist'
                            type: string
                          href: ...
                          id: ...
                          name: ...
                          uri: ...
                          external_urls: ...
                    type:
                      description: |
                        'Object type: track.'
                      type: string
                    name:
                      description: 'Track name.'
                      type: string
                    available_markets:
                      items:
                        type: string
                      type: array
                    disc_number: ...
                    duration_ms:
                      type: integer
                    href: ...
                    id: ...
                    explicit:
                      type: boolean
                    external_urls: ...
                    is_local: ...
                    is_playable: ...
                    linked_from: ...
                    preview_url: ...
                    restrictions:
                      type: object
                      properties:
                        reason:
                          type: string
                    track_number: ...
                    uri: ...
        '400':
          description: 'Invalid input'
        '500':
          description: 'API rate limit exceeded.'
```

Extracto de código 2.1: OAS Spotify.

El extracto de código 2.1 muestra un fragmento de la especificación OAS de la API de Spotify, uno de los sistemas usados para la evaluación de la propuesta presentada en este trabajo. La operación mostrada permite, a partir del id de un álbum, obtener la lista de canciones (Tracks) que lo componen. Para ejecutar esta operación, es necesario llamar al endpoint de la API con el path `'/album/{id}/tracks'` (línea 2) utilizando el método HTTP GET (línea 3). Además de información general sobre la operación, como su id y una descripción en lenguaje natural (líneas 4 y 5), la especificación indica los parámetros de entrada que pueden utilizarse (líneas 6-34), así como los distintos códigos de respuesta que la API puede devolver, incluyendo su formato (líneas 36-104).

Para esta operación, según indica la propiedad `required`, el único parámetro obligatorio es el id del álbum (líneas 7-12), un parámetro del path (según indica la propiedad `in`) de tipo string. Opcionalmente, es posible filtrar los resultados por país utilizando el parámetro `market` (líneas 13-18), además de gestionar la paginación con los parámetros `limit` (líneas 19-27) y `offset` (líneas 28-34).

Si el cliente introduce un id de un álbum que esté presente en la base de datos de Spotify, la API devolverá una respuesta en formato JSON etiquetada con un código 200 y con la estructura indicada en las líneas 39-100. Esta respuesta será de tipo objeto (línea 40) y contendrá propiedades con información general, como una URL e información sobre paginación y el número total de resultados (líneas 42-50), así como la propiedad de tipo array de objetos `items`, cada uno de los elementos de este array será una de las canciones que componen el álbum. Nótese que, para cada parámetro y campo de la respuesta, la especificación OAS indica su tipo de dato (string, boolean, integer, double, object o array).

2.2. Detección de invariantes

Un invariante es una propiedad que se cumple siempre en un punto o en una serie de puntos de la ejecución de un programa [44]. Por ejemplo, en una función que recibe como entrada una lista y le añade un elemento, un posible invariante sería que la lista devuelta siempre tiene un mayor tamaño que la que se pasa como parámetro. Existen dos corrientes principales para la detección automática de invariantes:

- **Análisis estático del código:** Consiste en detectar invariantes en un programa basándose únicamente en un análisis de su código fuente, sin ejecutarlo [41, 47]. Generalmente, este enfoque se caracteriza por generar invariantes sólidos y con una baja proporción de falsos positivos. Sin embargo, su principal desventaja reside en la limitada cantidad de invariantes que puede generar, que suelen ser muy poco específicos, dificultando el discernir distintos estados del programa.
- **Análisis de la ejecución del programa:** También conocido como *detección dinámica de invariantes*, consiste en detectar invariantes a partir de la ejecución de un programa [43, 48, 42]. Generalmente, este acercamiento permite detectar una mayor cantidad de invariantes que un análisis estático, con la desventaja de que los invariantes producidos se conocen como *invariantes potenciales* que se cumplen para las ejecuciones del programa bajo análisis, pero que podrían no cumplirse para otras ejecuciones en las que, por ejemplo, se usen otros valores para los parámetros de entrada, lo que resulta en un mayor riesgo de falsos positivos.

2.2.1. Daikon

Daikon [43] es una herramienta de detección dinámica de invariantes que detecta la existencia de invariantes potenciales estadísticamente justificados mediante la instrumentalización del programa que se está ejecutando. Daikon posee una serie de características que permiten su aplicación en toda clase de contextos:

- **Soporte para diferentes lenguajes de programación y otros formatos de datos:** Daikon puede detectar invariantes potenciales en programas escritos en distintos lenguajes de programación, además de soportar distintos formatos de salida que permiten, por ejemplo, generar assertions automáticamente para programas escritos en Java.

- **Salidas expresivas y personalizables:** La última versión de Daikon (lanzada en Noviembre de 2021) soporta un total de 143 invariantes distintos, algunos ejemplos son valores constantes ($x = a$), valores distintos de cero ($x \neq 0$), valores dentro de un rango ($a \leq x \leq b$), relaciones lineales ($y = ax + b$), ordenación ($x \leq y$) o subconjuntos ($x \in y$), entre otros. Cabe destacar que esta lista de invariantes es personalizable, ya que Daikon permite al usuario deshabilitar aquellos invariantes que puedan resultar en falsos positivos (o que no aporten información relevante en el contexto del problema a resolver), así como extender el sistema definiendo nuevos tipos de invariantes. Daikon también puede calcular invariantes sobre variables derivadas (que también pueden ser deshabilitadas y definidas por el usuario), como puede ser el tamaño (size) de las propiedades de tipo array.
- **Escalabilidad:** A pesar de soportar un amplio catálogo de invariantes y variables derivadas, Daikon es escalable a programas no triviales.
- **Filtrado de invariantes:** Al analizar la traza de un programa para detectar invariantes, existe el riesgo de reportar invariantes redundantes o que se han cumplido por pura coincidencia.
    - Para mitigar la posibilidad de reportar invariantes que hayan podido darse por casualidad, Daikon calcula, para cada invariante, una métrica de fiabilidad (confidence) en base al número de ocasiones en las que se ha encontrado el invariante, de manera que el invariante sea reportado si y solo si se ha superado un umbral de fiabilidad mínimo (Más detalles sobre el cálculo de esta métrica en la sección 4.2.1).
    - Con el objetivo de filtrar los invariantes redundantes, Daikon permite especificar que determinados invariantes están lógicamente implicados por otros (por ejemplo, el invariante `this.age is Positive` está lógicamente implicado por el invariante `this.age>=18`), de esta forma los invariantes redundantes no solo no serán reportados por Daikon, sino que tampoco son calculados, lo que optimiza considerablemente el rendimiento de la herramienta.

Instrumenter

Un instrumenter es una herramienta que convierte un conjunto de datos, generalmente una traza de ejecución de un programa escrito en un lenguaje de programación determinado, en un formato que puede ser utilizado como input para Daikon [7] (este proceso se conoce como instrumentalización). En concreto, un instrumenter genera los siguientes archivos:

- **DeclsFile:** Con extensión `.decls`, especifica la estructura de todos los puntos del programa (program points), entre los que se incluyen todos los objetos y las entradas y las salidas de todos los métodos y funciones.
- **DtraceFile:** Con extensión `.dtrace`, especifica los valores asignados a cada punto del programa definido en el DeclsFile.

Los intrumenters son específicos del lenguaje de programación. Actualmente, existen instrumenters de Daikon para Java, C++, .NET, Eiffel, Perl y archivos en formato

CSV [32], entre otros.

Ejemplo de uso

Para ilustrar el funcionamiento de Daikon, tomemos como ejemplo una clase `StackAr.java` que implemente una pila con un tamaño máximo (pasado como parámetro al crear la pila) y que tenga los atributos del Extracto de código 2.2 e implemente los métodos del Extracto 2.3.

```
1 Object[] theArray; // Array que contiene los elementos de la pila
2 int topOfStack; // Indice del elemento superior de la pila. -1 si la no contiene elementos
```

Extracto de código 2.2: Atributos de la clase StackAr

Supongamos que existe una clase `StackArTester.java` con un método main en el que se crean varias pilas y se ejecutan varias operaciones sobre ellas. En el contexto de detección de invariantes en pruebas software, podríamos entender la clase `StackAr.java` como el software que queremos probar y la clase `StackArTester.java` como un conjunto de pruebas.

```
1 void push(Object x) // Insertar elemento x
2 void pop() // Extrae el ultimo elemento insertado
3 Object top() // Devuelve el ultimo elemento insertado
4 Object topAndPop() // Extrae y devuelve el ultimo elemento insertado
5 boolean isEmpty() // Devuelve true si la pila está vacia, false en caso contario
6 boolean isFull() // Devuelve true si la pila está llena, false en caso contrario
7 void makeEmpty() // Elimina todos los elementos de la pila
```

Extracto de código 2.3: Métodos de la clase StackAr

Para detectar invariantes en este programa, comenzamos ejecutando un instrumenter sobre él para obtener un DeclsFile y un DtraceFile. Para instrumentalizar un programa escrito en Java, usaríamos el instrumenter Chicory [32], que crea program points para cada objeto y para la entrada y salida de cada método. Una vez disponemos de los archivos `.decls` y `.dtrace`, los usamos como inputs para Daikon, que nos devolverá una serie de invariantes para cada program point.

Por ejemplo, el Extracto de código 2.4 muestra algunos de los invariantes que se han cumplido en todo momento para todas las instancias del objeto `StackAr`. En concreto, Daikon ha detectado que el valor del atributo `topOfStack` debe ser mayor o igual que -1 (valdrá -1 cuando la pila esté vacía, condición que se cumple al menos en el momento de su creación) y menor estricto que el tamaño del array que representa la pila (al crear una nueva pila, el array tendrá un tamaño igual a la capacidad que se introduce como parámetro, con todos sus elementos siendo null).

```
1 this.topOfStack >= -1
2 this.topOfStack < this.theArray.length
```

Extracto de código 2.4: Invariantes del objeto StackAr

Por otra parte, el Extracto 2.5 muestra las precondiciones y postcondiciones del constructor de `StackAr.java`, que indican que el valor del parámetro que especifica el tamaño de la pila debe ser mayor o igual que cero (línea 2), que el tamaño del array que representa la pila será igual a la capacidad (línea 4), que el valor inicial del

atributo `topOfStack` será siempre -1 (línea 5) y que todos los elementos del array serán inicialmente nulos (línea 6).

```
1 // Precondiciones del constructor de StackAr
2 capacity >= 0
3 // Postcondiciones del constructor de StackAr
4 orig(capacity) == this.theArray.length
5 this.topOfStack == -1
6 this.theArray[] elements == null
```

Extracto de código 2.5: Invariantes del constructor de StackAr

Finalmente, el Extracto 2.6 contiene los invariantes detectados en la salida del método `isFull`. Daikon ha detectado automáticamente que ninguno de los atributos del objecto `StackAr` es modificado (líneas 1-3) y que el método devuelve `false` si la pila no está llena (línea 4), y `true` en caso contrario (línea 5).

```
1 this.theArray == orig(this.theArray)
2 this.theArray[] == orig(this.theArray[])
3 this.topOfStack == orig(this.topOfStack)
4 (return == false) <==> (this.topOfStack < this.theArray.length - 1)
5 (return == true) <==> (this.topOfStack == this.theArray.length - 1)
```

Extracto de código 2.6: Invariantes del método isFull

3. Estudio del estado del arte

Este trabajo se encuentra dentro del marco del testing automático de APIs RESTful. En los últimos años, han proliferado una gran cantidad de herramientas y técnicas orientadas a incrementar esta automatización, que pueden dividirse dentro de dos grandes categorías: de caja blanca y de caja negra.

Al requerir de acceso al código fuente del sistema, las técnicas de caja blanca son menos comunes. Arcuri [37] propuso una técnica que genera casos de prueba mediante algoritmos genéticos que busca maximizar la cobertura de código obtenida y el número de fallos encontrados.

Por otra parte, las técnicas de caja negra utilizan la información de la especificación OAS para generar casos de prueba, sin necesidad de acceder al código fuente, lo que permite que sean aplicables a cualquier API, independientemente de la tecnología usada para su implementación. Estas propuestas generan casos de prueba mediante la aplicación de diferentes técnicas como testing basado en modelos [52, 59] o en propiedades [49]. Otras propuestas buscan generar secuencias de operaciones de manera automática que se infieren analizando los nombres de los parámetros y los campos de la respuesta [38] o creando un grafo de dependencias entre operaciones [60]. También existen propuestas que generan valores de entrada de manera automática [34].

Sin embargo, tal y como señala un estudio comparativo publicado recientemente [51], todas estas herramientas se encuentran limitadas por el tipo de errores que pueden detectar. La Tabla 3.1 muestra los oráculos utilizados por las 10 principales herramientas de generación automática de pruebas para APIs RESTful que existen en la actualidad.

Herramienta	**Oráculos utilizados**	**Sitio web**
EvoMasterWB	Código de estado	[2]
EvoMasterBB	Código de estado	[2]
RESTler	Código de estado y verificaciones pre-definidas	[23]
RestTestGen	Código de estado y validación del formato de la respuesta	[24]
RESTest	Código de estado y validación del formato de la respuesta	[22]
Schemathesis	Código de estado y validación del formato de la respuesta	[25]
Dredd	Código de estado y validación del formato de la respuesta	[10]
Tcases	Código de estado	[28]
bBoxRT	Código de estado y análisis del comportamiento	[5]
APIFuzzer	Código de estado	[1]

Tabla 3.1: Oráculos utilizados por las principales herramientas de generación de pruebas para APIs RESTful. Fuente: [51]

Esta tabla revela que, aunque se han producido una gran cantidad de avances en generación automática de pruebas para este tipo de sistemas, actualmente sigue existiendo una importante limitación en lo que respecta a la generación automática de oráculos de prueba, lo que limita considerablemente la capacidad de detección de errores de estas técnicas. Algunos autores han propuesto técnicas basadas en el uso de pruebas metamórficas [58]. A pesar de su potencial y capacidad demostrada de detectar

errores en APIs comerciales como Spotify y YouTube, estas técnicas solo son aplicables a determinadas operaciones de búsqueda, además de requerir de especificaciones generadas manualmente.

Por otra parte, la detección automática de invariantes ha demostrado ser de gran utilidad en contextos tan variados como bases de datos relacionales [40], el mercado de valores [32], aplicaciones web [53], reparar vulnerabilidades en programas [62], sistemas ciberfísicos [61, 33] o robótica [50], entre otros. Además, los invariantes detectados por herramientas como Daikon han sido utilizados por varios autores para comprobar el impacto de la introducción de mutantes en un software [57], validación de inputs [54], y como feedback para fuzzers en casos en los que la cobertura de código resultaba insuficiente como métrica para validar la calidad de un conjunto de pruebas [46]. Sin embargo, esta técnica aún no ha sido aplicada en el contexto de generación de pruebas en APIs RESTful, a pesar de su potencial.

4. Descripción de la propuesta

El principal objetivo de este proyecto es desarrollar un sistema de generación automática de oráculos de prueba en el contexto de las pruebas de caja negra de APIs RESTful mediante un proceso de detección de invariantes potenciales.

Para ello, se ha desarrollado un instrumenter para Daikon que recibe como entrada la especificación OAS de una API RESTful (permitiendo extraer los formatos de entrada y salida de cada operación) y un conjunto de pruebas en formato CSV en el que cada instancia representa una prueba y contiene los valores usados para los parámetros de entrada y el cuerpo de la respuesta en formato JSON.

El instrumenter desarrollado procesa la especificación y las pruebas, generando una instrumentalización del conjunto de pruebas consistente en un DeclsFile y un DtraceFile que son utilizados como entradas para una versión modificada de Daikon que devuelve una lista de invariantes potenciales que pueden ser utilizados como oráculos. El procedimiento completo de la propuesta es el mostrado en la Figura 4.1.

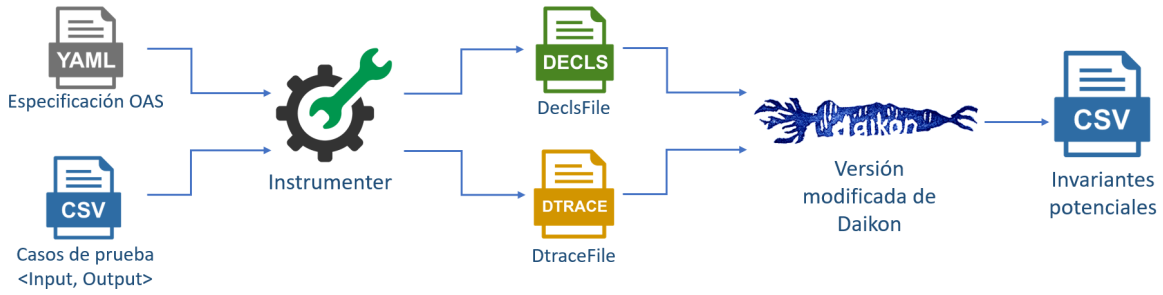

Figura 4.1: Diagrama del funcionamiento de la propuesta

En las siguientes secciones se detallan los detalles de diseño e implementación del instrumenter desarrollado, así como las modificaciones realizadas sobre el código de Daikon.

4.1. Instrumenter desarrollado

4.1.1. Generación del DeclsFile

Como se ha mencionado en la Sección 2.2, un DeclsFile especifica la estructura de todos los puntos de un programa, entre los que se incluyen las clases, los objetos, las entradas y salidas de todos los métodos y funciones que lo componen. Al estar trabajando en un contexto de pruebas de caja negra en el que no tenemos acceso al código fuente de la API, los únicos tipos de program points devueltos por el instrumenter serán aquellos que modelen el formato de las entradas (ENTER) y de las salidas (EXIT).

Para explicar de manera detallada la estructura de un DeclsFile y ejemplificar el proceso de instrumentalización sin mostrar información redundante, se utilizará como ejemplo una versión simplificada de la operación "Get Album Tracks" de la API de Spotify, mostrada en el Extracto 4.1.

```yaml
paths:
  '/albums/{id}/tracks':
    get:
      operationId: 'getAlbumTracks'
      parameters:
        - name: id
          description: |
            'The Spotify ID for the album'
          in: path
          required: true
          type: string
        - name: market
          description: |
            'An ISO 3166-1 alpha-2 country code'
          in: query
          required: false
          type: string
        - name: limit
          description: |
            'The maximum number of items to return'
          in: query
          required: false
          type: integer
      responses:
        '200':
          description: 'OK'
          schema:
            type: object
            properties:
              total:
                type: integer
              href:
                type: string
              items: # Array de objetos
                type: array
                items:
                  type: object
                  properties:
                    artists: # Array de objetos
                      type: array
                      items:
                        type: object
                        properties:
                          id:
                            type: string
                          name:
                            type: string
                    # Array de strings
                    available_markets:
                      type: array
                      items:
                        type: string
                    id:
                      type: string
                    name:
                      type: string
                    track_number:
                      type: integer
                    explicit:
                      type: boolean
                    linked_from: # Objeto anidado
                      type: object
                      properties:
                        id:
                          type: string
                        uri:
                          type: string
        '400':
          description: 'Invalid input'
          schema:
            type: object
            properties:
              status:
                type: integer
              message:
                type: string
```

Extracto de código 4.1: Versión reducida de la especificación OAS de Spotify.

Esta operación recibe parámetros de distinto tipo (`id` y `market` son de tipo string, mientras que `limit` es numérico) y que se envían en distintas partes de la petición HTTP (path y query). Por otra parte, hay dos posibles tipos de respuesta, cada una con su correspondiente código de estado (líneas 25 y 68).

El formato de la respuesta con código 200 ilustra las características más comunes de las respuestas devueltas por una API, al contener propiedades de distintos tipos de datos primitivos (numéricos, strings y booleans), arrays de objetos (propiedad `items` en las líneas 36-67, y propiedad `artists` en las líneas 39-47), arrays de tipos primitivos (`available_markets`, líneas 49-52, es un array de strings) y objetos anidados dentro de objetos (cada elemento del array de objetos `items`, contiene como propiedad un objeto `linked_from`, líneas 61-67). El Extracto de código 4.2 muestra un JSON que sigue el formato de respuesta de las respuestas con código 200.

```json
{
    "total": 14,
    "href": "https://api.spotify.com/v1/albums/4Em5W5HgYEvhpc2elrpKES/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",
            "track_number": 1,
            "explicit": false,
            "linked_from": {
                "id": "98cZPdKywnMGp8fnw2XTYU",
                "uri": "https://open.spotify.com/artist/98cZPdKywnMGp8fnw2XTYU"
            }
        }
    ]
}
```

Extracto de código 4.2: Ejemplo de respuesta en formato JSON.

El instrumenter desarrollado emplearía el OAS del Extracto 4.1 para generar un DeclsFile que contenga los formatos de entrada y salida de cada una de las operaciones. Dicho DeclsFile seguiría la estructura mostrada en el Extracto 4.3.

```
1  decl-version 2.0
2
3  ppt main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::ENTER
4  ...
5
6  ppt main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::ENTER
7  ...
8
9  ppt main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::ENTER
10 ...
11
12 ppt main.albums{id}tracks.getAlbumTracks&400(main.getAlbumTracks&Input):::ENTER
13 ...
14
15 ppt main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::EXIT1
16 ...
17
18 ppt main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::EXIT2
19 ...
20
21 ppt main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::EXIT3
22 ...
23
24 ppt main.albums{id}tracks.getAlbumTracks&400(main.getAlbumTracks&Input):::EXIT4
25 ...
```

Extracto de código 4.3: Estructura del DeclsFile.

El DeclsFile comienza especificando la versión del formato del archivo (línea 1), seguida de todos los program points que componen la API a instrumentalizar (en este caso, serán únicamente las entradas y las salidas). Existirá, como mínimo, un program point de entrada y de salida por cada código de respuesta, por motivos que se explicarán más adelante, cada propiedad de tipo array de objetos tendrá sus propios program point

de entrada y de salida.

Utilizar program points diferentes en función del código de respuesta permite, entre otras cosas, detectar invariantes en las entradas que nos indiquen las razones por las que se ha obtenido un determinado tipo de respuesta. Por ejemplo, si la API de Spotify está bien implementada, debería devolver un error de validación (código 400) si el usuario introduce valores inválidos para el parámetro numérico `limit` (0 o un valor negativo) o para los parámetros de tipo string `market` o `id` (Por ejemplo, con una longitud incorrecta). De esta forma, para el program point que modela las entradas para las que se ha devuelto una salida válida (`getAlbumTracks&200(main.getAlbumTracks&Input):::ENTER`), se obtendrán invariantes que modelarán este comportamiento (`input.limit >= 1`, `LENGTH(input.market)==2` y `LENGTH(input.id)==22`) que no necesariamente estarán presentes en las entradas de las llamadas que han devuelto un error de validación (`getAlbumTracks&400(main.getAlbumTracks&Input):::ENTER`). Los program points que representan entradas tendrán exactamente la misma estructura, diferenciándose únicamente por su nombre (línea 1), el Extracto 4.4 muestra un ejemplo de program point de entrada.

```
1  ppt main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::ENTER
2  ppt-type enter
3  variable input
4    var-kind variable
5    dec-type main.getAlbumTracks&Input
6    rep-type hashcode
7  variable input.id
8    var-kind field id
9    enclosing-var input
10   dec-type java.lang.String
11   rep-type java.lang.String
12 variable input.market
13   var-kind field market
14   enclosing-var input
15   dec-type java.lang.String
16   rep-type java.lang.String
17 variable input.limit
18   var-kind field limit
19   enclosing-var input
20   dec-type int
21   rep-type int
```

Extracto de código 4.4: Ejemplo de program point de entrada.

Cada program point comienza con la especificación de su nombre (línea 1) seguida del tipo de program point (línea 2). A continuación, se incluyen las declaraciones de todas las variables involucradas en el program point. En el Extracto 4.4 hay una variable de tipo objeto (`input`) que representa la entrada completa, con sus propiedades siendo los distintos parámetros de entrada (`input.id`, `input.market` e `input.limit`). En el contexto de nuestra propuesta, cada variable de Daikon puede contener los siguientes campos:

- `var-kind` <kind>: Especifica el tipo de variable. En nuestro contexto, sus posibles valores son:
  - `variable`: Empleado en las variables de tipo objeto de las entradas.
  - `return`: Empleado en las variables de tipo objeto de las salidas.

    - `field` <var-name>: Se utiliza en las variables de tipo primitivo que son propiedades de un objeto (que puede ser de tipo `variable` o `return`).
    - `array`: Utilizado en las variables que representan los elementos de un array.
- `enclosing-var` <enclosing-var-name>: Se utiliza si la variable es una propiedad de otra variable de tipo objeto, donde `enclosing-var-name` indica el nombre de dicho objeto. Este campo es requerido para las variables cuyo `var-kind` es `field` o `array`. En el Extracto superior, este campo está presente en las variables primitivas `input.id`, `input.market` e `input.limit`, que son propiedades de la variable de tipo objeto `input`.
- `dec-type` <language-declaration>: Es el nombre del tipo de variable usado en el programa sin instrumentalizar. Los nombres de los tipos primitivos deben seguir la nomenclatura de Java (`int`, `boolean`, `double` y `java.lang.String`), mientras que los nombres de las variables de tipo objeto son completamente libres, como es el caso de la variable `input` del Extracto 4.4 (líneas 3-6).
- `rep-type` <daikon-type>: Describe el tipo de datos de Daikon que será utilizado en el DtraceFile. En las variables primitivas, será igual que el dec-type, mientras que en las propiedades de tipo objeto (como `input`) se usará un hashcode. El valor de este campo puede ser `boolean`, `int`, `double`, `java.lang.String`, `hashcode` o un array de estos, lo que se indica con los caracteres "[]".
- `array` <dim>: Número de dimensiones de la variable, utilizado para las variables de tipo array. Sus posibles valores son 0 y 1. Su valor por defecto es 0, por lo que no estará presente si la variable no es de tipo array.

```
1  ppt main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::EXIT1
2  ppt-type subexit
3  variable input
4  ...
5  variable return
6    var-kind return
7    dec-type main.getAlbumTracks&Output&200
8    rep-type hashcode
9  variable return.total
10   var-kind field total
11   enclosing-var return
12   dec-type int
13   rep-type int
14 variable return.href
15   var-kind field href
16   enclosing-var return
17   dec-type java.lang.String
18   rep-type java.lang.String
19 variable return.items
20   var-kind field items
21   enclosing-var return
22   dec-type main.items[]
23   rep-type hashcode
24 variable return.items[..]
25   var-kind array
26   enclosing-var return.items
27   array 1
28   dec-type main.items[]
29   rep-type hashcode[]
```

Extracto de código 4.5: Ejemplo de program point de salida.

El Extracto de código 4.5 muestra el program point que especifica el formato de

la salida de la API cuando esta devuelve un código 200. Los program points de salida están numerados (línea 1) y son etiquetados como `subexit` (línea 2). Esto se debe a que, en otros lenguajes de programación, un método puede tener varias salidas en líneas distintas, que Daikon se encarga de unir al reportar los invariantes.

Al declarar un program point de salida, es necesario volver a incluir las variables de entrada al principio (las variables de entrada han sido omitidas en el Extracto 4.5 para evitar mostrar información redundante), lo que permite detectar invariantes que relacionan a variables de las entradas con variables de las salidas, como puede ser `input.limit >= size(return.items[])`.

Las variables de salida se encuentran a partir de la línea 5 del Extracto, donde se encuentra el objeto `return`, que contiene todos los campos de la respuesta como propiedades. Nótese que, en este caso, su var-kind es `return` en lugar de `variable`.

Como se mencionó al describir el formato de la respuesta del ejemplo ilustrativo, la propiedad `items` de la respuesta (`return.items` en el DeclsFile), es un array de elementos de tipo objeto, cada uno de los cuales representa una canción o track. Al especificar una propiedad de tipo array en un DeclsFile, es necesario emplear dos variables, una de tipo objeto representada mediante un hashcode que actúa como puntero (líneas 19-23) y una de tipo array que contenga los elementos del array (líneas 24-29).

A pesar de representarse mediante un hashcode (línea 23), el `var-kind` sigue siendo `field` (en lugar de `variable` o `return`), al tratarse de una propiedad anidada dentro del objeto que constituye la respuesta, que es su `enclosing-var` (línea 21). No obstante, si se tratase de un objeto anidado y no de un array, sería necesario usar `return` como `var-kind`.

La segunda variable, aquella que define los elementos del array, se declara como de tipo array (línea 25), por lo que es necesario especificar que su dimensionalidad no es 0 (línea 27), y su `enclosing-var` es el puntero (línea 26). Como `return.items` es un array de objetos, su representación será un array de hashcodes (línea 29).

Utilizando únicamente el hashcode de los elementos de los arrays de tipo objeto, no se pueden extraer invariantes relacionados con estos objetos, exceptuando aquellos relacionados con el tamaño del array, como `input.limit >= size(return.items[])`. Por este motivo, se define un nuevo program point cuyas variables son las propiedades del tipo de objeto que lo constituye.

```
1  ppt main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::EXIT2
2  ppt-type subexit
3  variable input
4  ...
5  variable return
6    var-kind return
7    dec-type main.getAlbumTracks&Output&200&items
8    rep-type hashcode
9  variable return.artists
10   var-kind field artists
11   enclosing-var return
12   dec-type main.artists[]
13   rep-type hashcode
14 variable return.artists[..]
15   var-kind array
16   enclosing-var return.artists
17   array 1
18   dec-type main.artists[]
19   rep-type hashcode[]
20 variable return.available_markets
21   var-kind field available_markets
22   enclosing-var return
23   dec-type java.lang.String[]
24   rep-type hashcode
25 variable return.available_markets[..]
26   var-kind array
27   enclosing-var return.available_markets
28   array 1
29   dec-type java.lang.String[]
30   rep-type java.lang.String[]
31 variable return.id
32 ...
33 variable return.name
34 ...
35 variable return.track_number
36 ...
37 variable return.explicit
38 ...
39 variable return.linked_from
40   var-kind return
41   enclosing-var return
42   dec-type main.getAlbumTracks&Output&200&items&linked_from
43   rep-type hashcode
44 variable return.linked_from.id
45   var-kind field id
46   enclosing-var return.linked_from
47   dec-type java.lang.String
48   rep-type java.lang.String
49 variable return.linked_from.uri
50   var-kind field uri
51   enclosing-var return.linked_from
52   dec-type java.lang.String
53   rep-type java.lang.String
```

Extracto de código 4.6: Segundo nivel de anidamiento de la salida.

Nótese que la jerarquía de anidamiento se representa utilizando el caracter "&" como separador para distinguir los distintos niveles (línea 1). Este program point contiene dos arrays, uno de objetos (líneas 9-19) y otro de strings (líneas 20-30). Aunque para el array de objetos será necesario definir un nuevo nivel de anidamiento (`200&items&artists`), esto no ocurrirá con el array de strings, ya que los elementos que lo componen pueden representarse en un DtraceFile sin recurrir a un nuevo nivel de anidamiento.

Además de estas propiedades de tipo array, el program point del Extracto 4.6 también contiene un objeto anidado (`return.linked_from`, líneas 39-53) con dos pro-

piedades primitivas (`return.linked_from.id` y `return.linked_from.uri`), para el que no es necesario crear un nuevo program point.

4.1.2. Generación del DtraceFile

Un DtraceFile contiene los valores asignados a cada una de las variables del programa (cada una de ellas perteneciente a un program point definido previamente en el DeclsFile) a lo largo de su ejecución. El Extracto 4.7 contiene un fragmento de un DtraceFile correspondiente a un program point de entrada. Al igual que ocurría en el DeclsFile, los program point de entrada de un mismo caso de prueba serán idénticos, diferenciándose únicamente por su nombre (línea 1).

```
1  main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::ENTER
2  input
3  1242334637
4  1
5  input.id
6  "4Em5W5HgYEvhpc2elrpKES"
7  1
8  input.market
9  "ES"
10 1
11 input.limit
12 1
13 1
```

Extracto de código 4.7: Ejemplo de program point de entrada en el DtraceFile.

Como puede observarse, cada representación de un program point tiene los siguientes componentes:

- **Nombre del program point** (línea 1).
- **Para cada variable:**
  - Nombre de la variable (líneas 2, 5, 8 y 11)
  - Valor de la variable (líneas 3, 6, 9 y 12):
    - Si la variable es numérica, su valor será una secuencia de dígitos que puede estar precedida por un signo menos. En el caso de las variables de tipo double, puede utilizarse un punto para indicar las cifras decimales.
    - Si la variable es un string, su valor debe estar entre comillas dobles.
    - Si la variable es un objeto, se representará mediante un hashcode, sin entrecomillar.
    - Si la variable es un array, los elementos que lo componen se separarán por espacios. Los elementos del array estarán entre los caracteres "[" y "]".
  - Bit modified (líneas 4, 7, 10 y 13): Su valor será 0 si la variable no ha sido asignada desde la última vez, y 1 en caso contrario. Los desarrolladores de Daikon recomiendan que su valor sea siempre 1 [7].

El Extracto de código 4.8 muestra la estructura que tendrá el DtraceFile completo para la ejemplificación de la propuesta, compuesta por un único caso de prueba. Nótese que cada EXIT debe ir acompañado por su correspondiente ENTER y que un mismo par de entradas y salidas puede repetirse en un mismo test si existe más de un elemento dentro de un array de objetos, como es el caso de los elementos de tipo artist (líneas 13-22).

```
1  main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::ENTER
2  ...
3
4  main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::EXIT1
5  ...
6
7  main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::ENTER
8  ...
9
10 main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::EXIT2
11 ...
12
13 main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::ENTER
14 ...
15
16 main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::EXIT3
17 ...
18
19 main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::ENTER
20 ...
21
22 main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::EXIT3
23 ...
```

Extracto de código 4.8: Estructura del DtraceFile.

El Extracto 4.9 muestra el valor del program point de salida para el primer nivel de anidamiento de la respuesta. Al igual que ocurría en el DeclsFile, al declarar un program point de salida es necesario volver a incluir las variables de entrada al principio (omitidas en el Extracto inferior para evitar mostrar información redundante).

```
1  main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::EXIT1
2  input
3  ...
4  return
5  2043815652
6  1
7  return.total
8  14
9  1
10 return.href
11 "https://api.spotify.com/v1/albums/4Em5W5HgYEvhpc2elrpKES/tracks?limit=1&market=ES"
12 1
13 return.items
14 1534143414
15 1
16 return.items[..]
17 [313805079]
18 1
```

Extracto de código 4.9: Primer nivel de anidamiento de la salida.

Además de dos variables primitivas (`return.total` y `return.href`), esta salida contiene una propiedad de tipo array de objetos (`return.items`), que se representa mediante dos variables de Daikon, un puntero de tipo objeto representado mediante un hashcode (líneas 13-15) y la variable que contiene los elementos del array, representados en este caso mediante hashcodes (líneas 16-18).

A continuación, se añadirá un par de program points (un ENTER y un EXIT) para cada elemento del array de objetos, como `return.items` solo contiene un elemento, solo se generarán un ENTER y un EXIT, este último representado en el Extracto 4.10. Este program point contiene el objeto anidado `return.linked_from` (líneas 31-39), el array de objetos `return.artists` (líneas 7-12), y el array de strings `return.available_markets` (líneas 13-18). En el caso de `return.available_markets`, al tratarse de un array con elementos primitivos no es necesario definir un nuevo nivel de anidamiendo para detectar invariantes (como `LENGTH(return.markets)==2`).

```
1  main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::EXIT2
2  input
3  ...
4  return
5  2043815652
6  1
7  return.artists
8  94025278
9  1
10 return.artists[..]
11 [1380183179 1380183148]
12 1
13 return.available_markets
14 1795074634
15 1
16 return.available_markets[..]
17 ["ES" "US" "JP"]
18 1
19 return.id
20 "0gFvkiT2afIcJwNxXQ7W51"
21 1
22 return.name
23 "Mrs. Robinson"
24 1
25 return.track_number
26 1
27 1
28 return.explicit
29 false
30 1
31 return.linked_from
32 1573436043
33 1
34 return.linked_from.id
35 "98cZPdKywnMGp8fnw2XTYU"
36 1
37 return.linked_from.uri
38 "https://open.spotify.com/artist/98cZPdKywnMGp8fnw2XTYU"
39 1
```

Extracto de código 4.10: Segundo nivel de anidamiento de la salida.

Finalmente, se generarán dos pares de entradas y salidas, uno para cada elemento de `return.artists` (Extracto de código 4.11).

```
main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::ENTER
...

main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::EXIT3
input
...
return
2043815652
1
return.id
"2CvCyf1gEVhI0mX6aFXmVI"
1
return.name
"Paul Simon"
1

main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::ENTER
...

main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::EXIT3
input
...
return
2043815652
1
return.id
"70cRZdQywnSFp9pnc2WTCE"
1
return.name
"Arthur Garfunkel"
1
```

Extracto de código 4.11: Tercer nivel de anidamiento de la salida.

El Extracto de código 4.12 muestra algunos de los invariantes que Daikon detectaría si lo ejecutasemos con un conjunto de pruebas como el de la ejemplificación de la propuesta.

```
1  ===========================================================================
2  main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::ENTER
3  LENGTH(input.id)==22
4  input.limit >= 1
5  LENGTH(input.market)==2
6  ===========================================================================
7  main.albums{id}tracks.getAlbumTracks&200(main.getAlbumTracks&Input):::EXIT
8  LENGTH(input.market)==2
9  return.href is Url
10 input.limit >= size(return.items[])
11 return.total >= size(return.items[])
12 return.total >= 1
13 ===========================================================================
14 main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::ENTER
15 ...
16 ===========================================================================
17 main.albums{id}tracks.getAlbumTracks&200&items(main.getAlbumTracks&Input):::EXIT
18 size(return.artists[]) >= 1
19 All the elements of return.available_markets[] have LENGTH=2
20 LENGTH(return.id)==22
21 return.track_number >= 1
22 LENGTH(return.href)==56
23 return.href is Url
24 LENGTH(return.linked_from.id)==22
25 return.linked_from.uri is Url
26 LENGTH(return.linked_from.uri)==54
27 ===========================================================================
28 main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::ENTER
29 ...
30 ===========================================================================
31 main.albums{id}tracks.getAlbumTracks&200&items&artists(main.getAlbumTracks&Input):::EXIT
32 LENGTH(return.id)==22
```

Extracto de código 4.12: Invariantes detectados para la ejemplificación de la propuesta.

4.2. Modificaciones realizadas sobre Daikon

En esta sección se detallan las distintas modificaciones llevadas a cabo sobre el código de Daikon para adaptarlo al contexto de generación de oráculos de pruebas para APIs RESTful. En concreto, se han realizado los siguientes cambios:

- Se han deshabilitado algunos de los tipos de invariantes disponibles por defecto por no aportar información relevante en el contexto actual, ya que simplemente podrían resultar en la aparición de falsos positivos (sección 4.2.1).
- Definición de nuevos tipos de invariantes aplicables en el contexto de las APIs RESTful (sección 4.2.1). Estos invariantes están basados en una publicación previa del autor de este trabajo en el que se realizó una evaluación con un conjunto de datos compuesto por 48 APIs reales [34].
- Definición de nuevos supresores (un supresor indica que un tipo de invariante está implicado por otro) para evitar que Daikon genere oráculos redundantes (sección 4.2.2).
- Supresión de variables derivadas que no aportan información valiosa en el contexto actual, evitando de esta forma que Daikon muestre información irrelevante (sección 4.2.3).

4.2.1. Modificaciones realizadas sobre el conjunto de invariantes

En concreto, se han añadido 22 invariantes nuevos propios del dominio de las APIs RESTful y se han deshabilitado 25 invariantes que no revelarían información relevante sobre el funcionamiento de la API y probablemente resultarían en la aparición de falsos positivos. Nuestra versión modificada de Daikon soporta un total de 140 invariantes, que pueden clasificarse dentro de las siguientes categorías:

- **Relaciones aritméticas:** Indican comparaciones del valor de las propiedades de tipo numérico. Por ejemplo, al realizar una búsqueda de álbumes en Spotify, el número de track de una canción debe ser mayor o igual que 1 (`return.track_number >= 1`).
- **Propiedades de arrays:** Indican que un array cumple con determinadas características. Por ejemplo, al buscar hoteles por id en Amadeus, el id de cada hotel devuelto debe encontrarse dentro del array de ids que se utiliza como parámetro de entrada (`return.hotelId in input.hotelIds[]`). Asimismo, el tamaño de la propiedad de tipo array que contiene los resultados (`data`) debe ser menor o igual que el tamaño de la lista de ids que se ha usado como parámetro (`size(input.hotelIds[]) >= size(return.data[])`).
- **Valores específicos:** Indican que una propiedad siempre tiene un valor o conjunto de valores fijos. Por ejemplo, en la API de GitHub, un repositorio puede ser público o privado (`return.visibility one of {"private", "public"}`).
- **Formatos específicos:** Indican que un campo de tipo string siempre sigue un formato concreto, como URLs, fechas o emails. Por ejemplo, el campo `Poster` de OMDb siempre debe ser de tipo URL (`return.Poster is Url`).

Invariantes deshabilitados

En este apartado se enumeran los invariantes de Daikon que han sido deshabilitados por no aportar información relevante en el contexto que nos ocupa. La decisión de eliminar estos invariantes se tomó tras una experimentación preliminar que reveló que generaban casi un 100 % de falsos positivos. Los invariantes deshabilitados son los siguientes [8]:

1. **IntNonEqual:** Indica que dos variables de tipo entero nunca son iguales. Se representa como `x != y`.
2. **FloatNonEqual:** Indica que dos variables de tipo double nunca son iguales. Se representa como `x != y`.
3. **StringNonEqual:** Indica que dos variables de tipo string nunca son iguales. Se representa como `x != y`.
4. **StringLessThan:** Indica que una variable de tipo string siempre tiene menos caracteres que una segunda variable del mismo tipo. Se representa como `x < y`.

5. **StringGreaterThan:** Indica que una variable de tipo string siempre tiene más caracteres que una segunda variable del mismo tipo. Se representa como `x > y`.
6. **StringLessEqual:** Indica que una variable de tipo string siempre tiene un número menor o igual de caracteres que una segunda variable del mismo tipo. Se representa como `x <= y`.
7. **StringGreaterEqual:** Indica que una variable de tipo string siempre tiene un número mayor o igual de caracteres que una segunda variable del mismo tipo. Se representa como `x >= y`.
8. **SeqSeqStringLessThan:** Indica que los elementos una variable de tipo array de strings siempre tienen menor longitud que los elementos de un segundo array del mismo tipo. Se representa como `x[] < y[] lexically`.
9. **SeqSeqStringGreaterThan:** Indica que los elementos una variable de tipo array de strings siempre tienen mayor longitud que los elementos de un segundo array del mismo tipo. Se representa como `x[] > y[] lexically`.
10. **SeqSeqStringLessEqual:** Indica que los elementos una variable de tipo array de strings siempre tienen una longitud menor o igual que los elementos de un segundo array del mismo tipo. Se representa como `x[] <= y[] lexically`.
11. **SeqSeqStringGreaterEqual:** Indica que los elementos una variable de tipo array de strings siempre tienen una longitud mayor o igual que los elementos de un segundo array del mismo tipo. Se representa como `x[] >= y[] lexically`.
12. **PairwiseStringLessThan:** Indica que la longitud de cada elemento i de una variable de tipo array de strings siempre es menor que la longitud del elemento i de un segundo array del mismo tipo. Se representa como `x[] < y[]`.
13. **PairwiseStringGreaterThan:** Indica que la longitud de cada elemento i de una variable de tipo array de strings siempre es mayor que la longitud del elemento i de un segundo array del mismo tipo. Se representa como `x[] > y[]`.
14. **PairwiseStringLessEqual:** Indica que la longitud de cada elemento i de una variable de tipo array de strings siempre es menor o igual que el elemento i de un segundo array del mismo tipo. Se representa como `x[] <= y[]`.
15. **PairwiseStringGreaterEqual:** Indica que la longitud de cada elemento i de una variable de tipo array de strings siempre es mayor o igual que el elemento i de un segundo array del mismo tipo. Se representa como `x[] >= y[]`.
16. **NumericInt$Divides:** Indica que una variable de tipo entero es divisible entre otra del mismo tipo. Se representa como `x % y == 0`.
17. **NumericFloat$Divides:** Indica que una variable de tipo double es divisible entre otra del mismo tipo. Se representa como `x % y == 0`.
18. **NonZeroFloat:** Indica que el valor de una variable de tipo double siempre es distinto de 0. Se representa como `x != 0`.
19. **NonZero:** Indica que el valor de una variable de tipo entero siempre es distinto de 0. Se representa como `x != 0`.

20. **EltNonZeroFloat:** Indica que todos los elementos de una variable de tipo array de doubles siempre son distintos de cero. Se representa como `x[] elements != 0`.

21. **EltNonZero:** Indica que todos los elementos de una variable de tipo array de enteros siempre son distintos de cero. Se representa como `x[] elements != 0`.

22. **NumericInt$Square:** Indica que el valor de una variable de tipo entero siempre es igual que el cuadrado del valor de otra variable del mismo tipo. Se representa como `x = y**2`.

23. **NumericFloat$Square:** Indica que el valor de una variable de tipo double siempre es igual que el cuadrado del valor de otra variable del mismo tipo. Se representa como `x = y**2`.

24. **LinearTernary:** Representa una relación linear entre tres variables de tipo entero. Se representa como `ax + by + cz + d = 0`.

25. **LinearTernaryFloat:** Representa una relación linear entre tres variables de tipo double. Se representa como `ax + by + cz + d = 0`.

Nuevos invariantes

En este apartado se especifican los nuevos invariantes definidos para el contexto de este trabajo, así como el procedimiento seguido para definir un nuevo invariante en Daikon. Para la definición de este conjunto de invariantes, se ha realizado un análisis de un conjunto de 48 APIs reales seleccionado de forma sistemática por el autor para una publicación anterior [34]. Los invariantes definidos son los siguientes:

1. **IsUrl:** Indica que el valor de una variable de tipo string siempre es una URL. Se representa como `x is Url`.

2. **SequenceStringElementsAreUrl:** Indica que todos los elementos de un array de strings son URLs. Se representa como `All the elements of x are URLs`.

3. **FixedLengthString:** Indica que el valor de una variable de tipo string siempre tiene una longitud fija n. Se representa como `LENGTH(x)==n`.

4. **SequenceFixedLengthString:** Indica que todos los elementos de un array de strings tienen una longitud fija n. Se representa como `All the elements of x have LENGTH=n`.

5. **IsNumeric:** Indica que los caracteres de una variable de tipo string siempre son numéricos. Se representa como `x is Numeric`.

6. **SequenceStringElementsAreNumeric:** Indica que los caracteres de todos los elementos de un array de strings son siempre numéricos. Se representa como `All the elements of x are Numeric`.

7. **IsEmail:** Indica que el valor de una variable de tipo string siempre es un email. Se representa como `x is Email`.

8. **SequenceStringElementsAreEmail:** Indica que todos los elementos de un array de strings son emails. Se representa como `All the elements of x are emails`.

9. **IsDateDDMMYYYY:** Indica que el valor de una variable de tipo string es siempre una fecha que sigue el formato DD/MM/YYYY (el separador puede ser el caracter "/" o "-"). Se representa como `x is a Date. Format: DD/MM/YYYY`.

10. **SequenceStringElementsAreDateDDMMYYYY:** Indica que todos los elementos de un array de strings son fechas que siguen el formato DD/MM/YYYY. Se representa como `All the elements of x are dates. Format: DD/MM/YYYY`.

11. **IsDateMMDDYYYY:** Indica que el valor de una variable de tipo string es siempre una fecha que sigue el formato MM/DD/YYYY (el separador puede ser el caracter "/" o "-"). Se representa como `x is a Date. Format: MM/DD/YYYY`.

12. **SequenceStringElementsAreDateMMDDYYYY:** Indica que todos los elementos de un array de strings son fechas que siguen el formato MM/DD/YYYY. Se representa como `All the elements of x are dates. Format: MM/DD/YYYY`.

13. **IsDateYYYYMMDD:** Indica que el valor de una variable de tipo string es siempre una fecha que sigue el formato YYYY/MM/DD (el separador puede ser el caracter "/" o "-"). Se representa como `x is a Date. Format: YYYY/MM/DD`.

14. **SequenceStringElementsAreDateYYYYMMDD:** Indica que todos los elementos de un array de strings son fechas que siguen el formato YYYY/MM/DD. Se representa como `All the elements of x are dates. Format: YYYY/MM/DD`.

15. **IsHour:** Indica que el valor de una variable de tipo string es siempre una hora en formato de 24 horas. Se representa como `x is Hour: HH:MM 24-hour format, optional leading 0`.

16. **SequenceStringElementsAreHour:** Indica que todos los elementos de un array de strings son horas en formato de 24 horas. Se representa como `All the elements of x are Hours: HH:MM 24-hour format, optional leading 0`.

17. **IsHourAMPM:** Indica que el valor de una variable de tipo string es siempre una hora en formato de 12 horas. Se representa como `x is Hour: HH:MM 12-hour format, optional leading 0`.

18. **SequenceStringElementsAreHourAMPM:** Indica que todos los elementos de un array de strings son horas en formato de 12 horas. Se representa como `All the elements of x are Hours: HH:MM 12-hour format, optional leading 0, mandatory meridiems (AM/PM)`.

19. **IsHourWithSeconds:** Indica que el valor de una variable de tipo string es siempre una hora en formato de 24 horas, con los segundos incluidos. Se representa como `x is Hour: HH:MM:SS 24-hour format with optional leading 0`.

20. **SequenceStringElementsAreHourWithSeconds:** Indica que todos los elementos de un array de strings son horas en formato de 24 horas, con los segundos incluidos. Se representa como `All the elements of x are Hours: HH:MM:SS 24-hour format with optional leading 0`.

21. **IsTimestampYYYYMMHHThhmmssmm:** Indica que el valor de una variable de tipo string es siempre un timestamp. Se representa como `x is Timestamp. Format: YYYY-MM-DDTHH:MM:SS.mmZ (Miliseconds are optional)`.

22. **SequenceStringElementsAreTimestampYYYYMMHHThhmmssmm:** Indica que todos los elementos de un array de strings son timestamps. Se representa como `All the elements of x are Timestamps. Format: YYYY-MM-DDTHH:MM:SS.mmZ (Miliseconds are optional)`.

Definición de nuevos invariantes

Los invariantes de Daikon pueden ser unarios (solo involucran a una variable), binarios (involucran a dos variables) o ternarios (involucran a tres variables). Añadir un nuevo invariante a Daikon requiere escribir una clase Java. Al iniciar su ejecución, Daikon genera instancias de todos los posibles invariantes que pueden generarse. Si durante el análisis del DtraceFile, una muestra de un program point contradice a un invariante, este es destruido. Al final de la ejecución, se calcula la fiabilidad (confidence) de cada invariante restante. Si esta métrica supera un mínimo, el invariante es reportado al usuario.

De esta forma, para que un invariante sea reportado deben cumplirse dos condiciones: (1) El invariante se cumple para todas las instancias del program point en el DtraceFile (i.e., no existe ningún contraejemplo) y (2) la fiabilidad del invariante supera un valor mínimo.

A continuación, se muestra un ejemplo de un invariante definido en este trabajo (`IsUrl`) para ejemplificar la estructura que debe seguir un invariante. A la hora de definir la clase que define al invariante, esta debe extender a una subclase de `Invariant`. En concreto, el invariante `IsUrl` extiende a `SingleString` (línea 1 del Extracto 4.13), la clase usada para definir invariantes unarios (aquellos que solo involucran a una variable) para variables de tipo string.

Es necesario implementar un constructor y un método estático `get_proto` (Extracto 4.13). El método `get_proto` se utiliza para instanciar el invariante dentro de un program point (`PptSlice` es la clase que contiene información sobre el program point).

```
1  public class IsUrl extends SingleString {
2
3      private IsUrl(PptSlice ppt){ super(ppt); }
4
5      private @Prototype IsUrl() { super(); }
6
7      private static @Prototype IsUrl proto = new @Prototype IsUrl();
8
9      // Returns the prototype invariant
10     public static @Prototype IsUrl get_proto() { return proto; }
```

Extracto de código 4.13: Constructor y método get_proto

Para cada instancia de un program point del DtraceFile, Daikon comprueba que el invariante sigue cumpliéndose, utilizando para ello los métodos `add_modified` y `check_modified`, que devuelven un `InvariantStatus`, cuyo valor puede ser `NO_CHANGE`

(el invariante sigue considerándose válido), `FALSIFIED` (El invariante es destruido) y `WEAKENED` (la condición del invariante es debilitada).

- `check_modified` recibe una muestra y devuelve si esta es consistente con el invariante. En el caso de `IsUrl`, el invariante sigue considerándose válido si el valor de la variable sigue una expresión regular que represente una URL, siendo destruido en caso contrario.
- `add_modified` cambia el estado del invariante si es necesario. Si el invariante no tiene un estado (como es el caso de `IsUrl`), este método simplemente llama a `check_modified`. Se dice que un invariante tiene estado si este almacena algún tipo información adicional.

Por ejemplo, el invariante `OneOf` indica que una variable solo puede tener un conjunto de valores claramente definidos (el tamaño máximo de dicho conjunto es configurable), se representa como `x one of {c1, c2, c3}`. El estado de este invariante almacena dicho conjunto de valores de manera que, si durante la ejecución de Daikon se detecta un valor adicional, se llama al método `add_modified`, que modifica el estado añadiendo dicho valor al conjunto e indicando que el estado del invariante se ha debilitado (i.e., devuelve el InvariantStatus `WEAKENED`).

```
1   public InvariantStatus check_modified(String v, int count) {
2       Pattern pattern = Pattern.compile(urlRegex);
3       Matcher matcher = pattern.matcher(v);
4
5       if (matcher.matches()) {
6           return InvariantStatus.NO_CHANGE;
7       }
8       return InvariantStatus.FALSIFIED;
9   }
10
11  public InvariantStatus add_modified(String v, int count) { return check_modified(v, count); }
```

Extracto de código 4.14: Métodos check_modified y add_modified

El método `format_using` devuelve un string con la representación del invariante que será reportada al usuario (Extracto de código 4.15).

```
1   public String format_using(@GuardSatisfied IsUrl this, OutputFormat format) {
2           return var().name() + " is Url";
3       }
```

Extracto de código 4.15: Método format_using

El método `isSameFormula` (Extracto 4.16) devuelve true si dos invariantes representan la misma fórmula matemática, sin tener en cuenta contexto como los nombre de las variables, el valor de la confidencialidad o el número de muestras. En el caso de un invariante unario como `IsUrl`, basta con comprobar que los dos invariantes pertenecen a la misma clase.

```
1   public boolean isSameFormula(Invariant other) {
2           if (other instanceof IsUrl) {
3               return true;
4           }
5           return false;
6       }
```

Extracto de código 4.16: Método isSameFormula

Finalmente, el método `computeConfidence` calcula la fiabilidad de un invariante. El valor de la fiabilidad de un invariante puede encontrarse entre 0 y 1, y debe alcanzar un mínimo (0,99 por defecto) para ser reportado al usuario. Por ejemplo, el método del Extracto 4.17 asume que cada variable tiene una posibilidad del 10 % de ser una URL por casualidad. De esta forma, un conjunto de n valores tiene una posibilidad de $(0{,}1)^n$ de que todos sean URLs por casualidad.

```
1   protected double computeConfidence() {
2       return 1 - Math.pow(.1, ppt.num_samples());
3   }
```

Extracto de código 4.17: Método computeConfidence

4.2.2. Supresores de invariantes

Daikon permite definir supresores para evitar reportar información redundante y hacer la detección de invariantes más eficiente. Si un invariante A implica un invariante B, B no es instanciado ni revisado mientras A siga cumpliéndose.

Un invariante supresor (definido mediante la clase `NISuppressor`) pertenece a un conjunto de invariantes (`NISuppression`) que implican (y por lo tanto suprimen) a un invariante suprimido (definido mediante la clase `NISuppressee`). El conjunto de todos los invariantes supresores que suprimen a un invariante suprimido se almacenan en la clase `NISuppressionSet`.

Dentro de los invariantes definidos para este trabajo, algunos de los invariantes relacionados con fechas (`IsDateDDMMYYYY`, `IsDateMMDDYYYY` e `IsDateYYYYMMDD`) tienen una longitud fija, por lo que suprimen al invariante `FixedLengthString`. Para representar esto, definimos un método `get_ni_suppressions()` (Extracto 4.18) dentro de la clase del invariante a suprimir (en este caso, `FixedLengthString`). Dentro de dicho método, definimos los supresores, que son devueltos como un conjunto.

```
1   public @NonNull NISuppressionSet get_ni_suppressions() {
2       if (suppressions == null) {
3
4           NISuppressee suppressee = new NISuppressee(FixedLengthString.class, 1);
5
6           // suppressor definitions (used in suppressions below)
7           NISuppressor isDateMMDDYYYY = new NISuppressor(0, IsDateMMDDYYYY.class);
8           NISuppressor isDateDDMMYYYY = new NISuppressor(0, IsDateDDMMYYYY.class);
9           NISuppressor isDateYYYYMMDD = new NISuppressor(0, IsDateYYYYMMDD.class);
10
11
12          suppressions = new NISuppressionSet(
13                  new NISuppression[]{
14
15                          new NISuppression(isDateMMDDYYYY, suppressee),
16                          new NISuppression(isDateDDMMYYYY, suppressee),
17                          new NISuppression(isDateYYYYMMDD, suppressee),
18
19                  });
20      }
21      return suppressions;
22  }
```

Extracto de código 4.18: Método get_ni_suppressions

De esta forma, si el valor de una variable es siempre una fecha en uno de los formatos que constituyen el conjunto de supresores, Daikon no calculará (ni reportará) el estado de `FixedLengthString`, a menos que en un program point futuro el valor de la variable deje de seguir el formato de la fecha, pero siga manteniendo una longitud fija. En ese caso, el invariante supresor será destruido y el invariante suprimido dejará de considerarse como tal, reportándose al usuario si no se incumple en ningún program point posterior del DtraceFile.

4.2.3. Supresión de variables derivadas

Una variable derivada es una expresión que no aparece directamente en el programa instrumentalizado, pero que Daikon utiliza como variable para la detección de invariantes. Por ejemplo, para las variables de tipo array, Daikon genera una variable derivada cuyo valor sea su tamaño, permitiendo detectar invariantes como `input.limit >= size(return.items)`.

Sin embargo, existen variables derivadas que no aportan ningún tipo de información relevante, este es el caso de la variable derivada `orig()`, que indica el valor de una variable en el program point de entrada y se utiliza para realizar comparaciones con el valor de esa misma variable en el program point de salida. En el contexto de las pruebas de caja negra de APIs RESTful, el valor de los parámetros de entrada no cambia en ningún momento, por lo que Daikon detectaría únicamente invariantes de igualdad entre el valor del parámetro en la entrada y en la salida (`input.limit == orig(input.limit)`) que siempre se cumplirán. Por este motivo, esta variable derivada ha sido deshabilitada.

5. Validación

5.1. Preguntas de investigación

Con esta evaluación, se pretende responder a las siguientes preguntas de investigación (*Research Questions*, RQs):

- **RQ1:** *¿Cómo de efectiva es la propuesta presentada generando oráculos de prueba?* Este es el principal objetivo de este trabajo. Para contestar a esta pregunta, se ha analizado la precisión (en términos de verdaderos positivos y falsos positivos) de la propuesta para generar invariantes que modelen correctamente el comportamiento esperado de la API y por tanto puedan emplearse como oráculos de prueba. También se han analizado las principales causas por las que se generan los oráculos que han sido clasificados como falsos positivos.
- **RQ2:** *¿Cuál es el impacto del tamaño del conjunto de pruebas en la calidad del conjunto de oráculos generados?* La precisión de los invariantes potenciales inferidos de forma dinámica depende principalmente de la calidad y la exhaustividad de los casos de prueba utilizados. Con esta pregunta se pretende determinar el número aproximado de llamadas necesarias para generar un conjunto de oráculos que contenga el menor porcentaje posible de falsos positivos, analizando en qué medida aumenta la precisión obtenida al aumentar el número de casos de pruebas usados para la generación de invariantes.
- **RQ3:** *¿Cuál es la capacidad de la propuesta para la detección de errores?* Para responder a esta pregunta, se ha analizado la cantidad de invariantes detectados que modelan un comportamiento ilógico o inconsistente y que, por tanto, revelan la existencia de bugs en la implementación o de inconsistencias en la documentación.

5.2. Diseño experimental

Mediante la ejecución del experimento propuesto en esta sección, se busca responder a las preguntas de investigación planteadas anteriormente, evaluando la capacidad de la propuesta presentada para generar oráculos de prueba y detectar errores a partir de un conjunto de casos de prueba de un tamaño determinado.

Para la evaluación, se han utilizado 8 operaciones de 6 APIs comerciales distintas, mostradas en la Tabla 5.1. Para cada una de estas operaciones, se ha generado automáticamente un conjunto de pruebas utilizando el framework RESTest [52]. En este experimento, cada caso de prueba está formado por una única llamada a la API. En concreto, se han generado casos de prueba hasta obtener un total de 1000 llamadas exitosas (i.e., devuelven un código 2XX) para cada operación, este conjunto de 1000 llamadas será el que compondrá el conjunto de pruebas completo usado para detectar

API	API	Documentación
Amadeus Hotel	Find hotel offers	[4]
GitHub	List organization repositories	[12]
OMDb	By ID or Title	[3]
OMDb	By Search	[3]
Spotify	Create Playlist	[26]
Spotify	Get Album tracks	[27]
Yelp	Search businesses	[30]
YouTube	List videos	[31]

Tabla 5.1: Operaciones usadas para la evaluación

invariantes en cada operación (aunque, como se explicará más adelante, también se utilizarán subconjuntos de menor tamaño).

Muchas de las operaciones que componen el conjunto de evaluación son operaciones de búsqueda de recursos a partir de una serie de parámetros de filtrado, algunos de los filtros de estas operaciones (como ocurre en el caso de Yelp) pueden ser muy restrictivos, resultando en un gran porcentaje de respuestas que contienen 0 resultados y que aportan poca o ninguna información sobre el comportamiento de la API. Por estas razones, para la generación del conjunto de 1000 llamadas, se ha garantizado que al menos el 90 % de ellas contengan al menos 1 resultado.

Este conjunto de llamadas se ha utilizado, junto con la especificación OAS de cada API, como entrada para nuestra propuesta, resultando en un conjunto de invariantes potenciales. Con el objetivo de responder a RQ1 y RQ3, se han analizado, de manera manual, los invariantes detectados en cada operación, clasificándolos como verdaderos positivos, falsos positivos o como bugs/inconsistencias. Consideramos un invariante como un falso positivo si este modela un comportamiento que ha estado presente en el conjunto de llamadas usadas como entrenamiento, pero que no necesariamente se cumplirá siempre. Por ejemplo, consideramos como un falso positivo que la duración en milisegundos de una canción de Spotify sea mayor que su número de artistas (`return.duration_ms >size(return.artists[])`). Como parte del análisis, se han determinado las principales causas detrás de la aparición de estos falsos positivos.

Para responder a RQ2, se ha dividido cada conjunto de 1000 llamadas en conjuntos de 1000, 500, 100 y 50 llamadas, donde cada conjunto es un subconjunto del anterior. Para cada uno de estos conjuntos se ha calculado la precisión obtenida, determinando de esta forma el tamaño más apropiado que debe tener el conjunto de pruebas utilizado como entrada para la propuesta, además de analizar la evolución del proceso de detección de invariantes en términos de precisión a medida que se incrementa de manera gradual el número de casos de prueba.

Tanto RESTest como nuestra propuesta requieren de una especificación OAS para la generación de casos de prueba y la generación de oráculos, respectivamente. Para aquellas APIs que no ofrecen acceso a dicha especificación, se ha descargado la versión disponible en el repositorio APIs.guru [36] (Spotify y YouTube) y, en caso de que tampoco estuvieran disponibles en este repositorio, se han generado a mano a partir de la documentación web (OMDb y Yelp).

RESTest ofrece distintas opciones para configurar el proceso de generación de

casos de prueba, entre los que se encuentra la asignación de valores a los parámetros de entrada. Si bien es cierto que RESTest ofrece la posibilidad de generar estos valores automáticamente [34], para esta evaluación se han utilizado diccionarios de datos generados manualmente con el objetivo de evitar posibles sesgos relacionados con la generación de valores inválidos o poco variados. Al configurar estos valores manualmente, se ha buscado que sean lo más variados posible. Por ejemplo, para los ids de hoteles de la API de Amadeus, se han utilizado hoteles pertenecientes a varios países localizados en distintos continentes para que el comportamiento de la API sea lo más variado posible.

La API de OMDb no sigue las buenas prácticas de REST [56] ya que, en lugar de devolver un código 4XX cuando se introducen valores inválidos, devuelve una respuesta con un código 200 que contiene únicamente la propiedad de tipo objeto `error`. En el caso de esta API, se han considerado como exitosas aquellas respuestas que no contenían esta propiedad.

5.3. Resultados experimentales

En esta sección se describen los resultados experimentales relativos a la efectividad de la propuesta para generar oráculos de prueba (*RQ1*), al impacto del tamaño del conjunto de pruebas en la calidad de los oráculos generados (*RQ2*) y a la capacidad de detección de errores de la propuesta (*RQ3*).

5.3.1. Efectividad para la generación de oráculos de prueba

La Tabla 5.2 muestra el porcentaje de oráculos válidos generados en cada conjunto de llamadas válidas (columna "P (%)"), así como el total de invariantes potenciales detectados (columna "I"). Para responder a RQ1, se han analizado los resultados obtenidos por la propuesta al procesar el conjunto de pruebas completo (1000 llamadas).

API - Operation	50 API calls			100 API calls			500 API calls			1000 API calls		
	I	P (%)	Bug	I	P (%)	Bug	I	P (%)	Bug	I	P (%)	Bug
Amadeus Hotel - Find hotel offers	93	60.2	0	97	57.7	1	101	57.4	1	99	59.6	1
GitHub - List organization repositories	106	70.1	9	95	72.6	9	92	75	9	94	74.5	9
OMDb - By ID or Title	21	70	1	17	82.4	1	16	87.5	1	17	82.4	1
OMDb - By Search	6	100	1	6	100	1	7	100	1	6	100	1
Spotify - Create Playlist	28	100	0	28	100	0	28	100	0	28	100	0
Spotify - Get Album tracks	51	78.4	0	52	78.8	0	51	80.4	0	51	80.4	0
Yelp - Search businesses	25	32	0	25	32	0	22	36.4	0	19	42.1	0
YouTube - List videos	171	59.1	0	182	56.6	0	194	55.7	0	187	57.2	0
TOTAL	**501**	**65.3**	**11**	**502**	**64.7**	**12**	**511**	**65.2**	**12**	**501**	**66.5**	**12**

Tabla 5.2: Resultados experimentales. I="Invariantes", P="Precisión"

La propuesta ha obtenido una precisión total del 66.5 % (333 de los 501 invariantes detectados son oráculos válidos). Esta precisión ha oscilado entre un 42.1 % en la API de Yelp y un 100 % en las operaciones "Create playlist" y "By Search" de las API de Spotify y OMDb, respectivamente.

En total, se han detectado 501 invariantes distintos. El número de invariantes detectados por operación es directamente proporcional al número de parámetros y campos de la respuesta. El número de invariantes ha oscilado entre 6 en la operación "By Search" de OMDb y 187 en la API de YouTube.

La propuesta ha sido capaz de generar una gran cantidad de oráculos que modelan múltiples aspectos del comportamiento de la API, algunos de los cuales podrían pasar desapercibidos incluso para usuarios y desarrolladores conocedores del dominio de la API. A continuación, se presentan algunos ejemplos de oráculos generados para cada una de las operaciones utilizadas para la evaluación:

- **Amadeus Hotel - Find hotel offers:** Esta operación recibe como parámetro una lista de ids de hoteles y devuelve una lista de objetos de tipo hotel, cada uno de los cuales contiene una lista de ofertas. La propuesta ha detectado automáticamente que el número de hoteles devueltos es menor o igual que el número de ids pasados por parámetro (`size(input.hotelIds[]) >= size(return.data[])`) y que el id de cada uno de los objetos de tipo hotel debe encontrarse dentro de la lista de ids utilizada como parámetro (`return.hotel.hotelId in input.hotelIds[]`). También se han detectado relaciones entre los parámetros de entrada y los campos de la salida (e.g., `input.checkInDate == return.checkInDate`) y que ciertos campos de tipo string deben seguir un formato determinado (e.g., `LENGTH(return.hotel.hotelId)==8`, `return.checkOutDate is a Date. Format: YYYY/MM/DD`, `return.commission.percentage is Numeric`).
- **GitHub - List organization repositories:** La propuesta ha sido capaz de detectar los distintos valores que pueden tomar los parámetros de tipo enumerado de la respuesta (e.g., `return.visibility one of {"private", "public"}`), así como los formatos que deben seguir ciertos campos de tipo string (e.g., `return.owner.url is Url`, `return.created_at is Timestamp. Format: YYYY-MM-DDTHH:MM:SS.mmZ`).
- **OMDb - By ID or Title:** Los oráculos generados para esta operación establecen relaciones de igualdad entre los parámetros de entrada y los campos de la salida (e.g., `input.i == return.imdbID`), además de especificar el formato que deben seguir ciertos campos de tipo string (e.g., `return.Season is Numeric`).
- **OMDb - By Search:** Al igual que ha ocurrido con la otra operación de esta API, la propuesta ha identificado principalmente relaciones de igualdad entre las entradas y salidas (e.g., `return.totalResults is Numeric`) e invariantes que especifican el formato de algunos campos de la salida (`input.type == return.Type`).
- **Spotify - Create Playlist:** Para esta operación, se ha detectado, entre otros invariantes, que la lista de reproducción debe pertenecer al usuario que la ha creado (`input.user_id == return.owner.id`), que el objeto devuelto debe ser de tipo "playlist" (`return.type == "playlist"`) y que la lista de canciones debe estar vacía al crear la lista de reproducción (`return.tracks.items[] == []`).
- **Spotify - Get Album tracks:** En esta operación, se han detectado principalmente invariantes relacionados con la paginación (e.g., `input.limit == return.limit`, `input.limit >= size(return.items[])`) y con el formato de

algunos campos de tipo string y array de strings (e.g., `return.href is Url`, `All the elements of return.available_markets[] have LENGTH=2`).

- **Yelp - Search businesses:** De nuevo, se han detectado invariantes relacionados con el uso de paginación (e.g., `input.limit >= size(return.businesses[])`), con valores de campos de tipo enumerado (e.g., `return.transactions[] elements one of {"delivery", "pickup", "restaurant_reservation"}`) y con el formato de campos de tipo string (e.g., `LENGTH(return.location.country)==2`).
- **YouTube - List videos:** La aplicación de la propuesta sobre esta operación ha permitido detectar automáticamente los posibles valores de algunas de las características de los vídeos (e.g., `return.contentDetails.dimension one of {"2d", "3d"}`), el comportamiento de la paginación (e.g., `input.maxResults == return.pageInfo.resultsPerPage`) y el formato de algunos campos de la respuesta (e.g., `return.liveStreamingDetails.concurrentViewers is Numeric`).

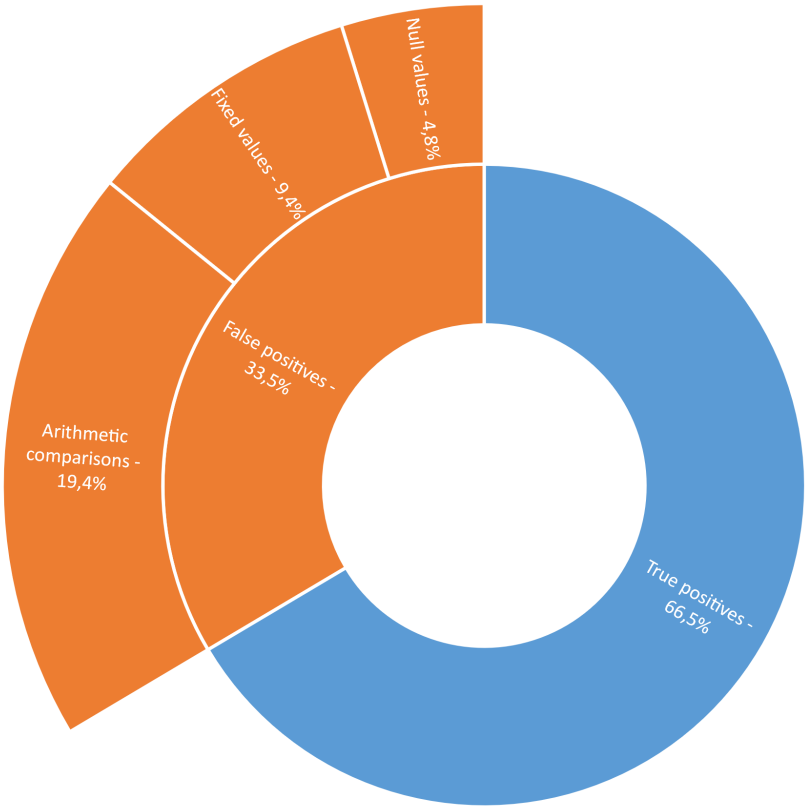

Figura 5.1: Distribución de los Falsos Positivos

Tras realizar un análisis de los invariantes clasificados como falsos positivos, estos se han catalogado dentro de uno de los siguientes tipos:

- **Valores nulos:** Se dan cuando, para todos los casos de prueba, un determinado campo de la respuesta es siempre nulo. Este ha sido el caso, por ejemplo, de algunos campos opcionales devueltos por la operación "Find hotel offers" de la API de Amadeus. El creador de una oferta devuelta por esta API puede especificar una descripción de la misma, sin embargo, su valor siempre ha sido nulo. Aunque

este tipo de invariantes puede resultar en falsos positivos, pueden ser útiles para encontrar limitaciones en el conjunto de pruebas usado para la detección de invariantes, además de permitir la identificación de campos en desuso que revelen la existencia de un error en la API (como ha sido el caso de GitHub, sección 5.3.3).

- **Valores fijos:** Se dan cuando se detectan invariantes que establecen que un determinado campo de la respuesta solo puede tener un valor o un conjunto de valores específicos, a pesar de que la API permita más. Este tipo de falsos positivos se han detectado, entre otras, en la operación "By ID or Title" de OMDb, donde solo 1 de los 1000 casos de prueba contenía un valor para el campo `Production` de la respuesta, por lo que la propuesta ha identificado este como su único valor válido (`return.Production == "Crappy World Films"`).

  Estos falsos positivos aparecen principalmente en aquellos casos en los que la API no ha devuelto todos los posibles valores de un campo de tipo enumerado a lo largo de todas las llamadas que componen el conjunto de pruebas. Por ejemplo, en la API de Amadeus Hotel, el valor del campo `type` del objeto `rateFamilyEstimated` puede ser "C", "P" o "N". Sin embargo, ninguna de las respuestas contenía el valor "N", resultando en un falso positivo (`return.rateFamilyEstimated.type one of { "C", "P" }`). Al igual que con los valores nulos, este tipo de falsos positivos pueden utilizarse para encontrar carencias en el conjunto de pruebas o para identificar bugs en la implementación de la API (como ha ocurrido en OMDb, sección 5.3.3).

- **Comparaciones aritméticas:** Aparecen cuando determinadas comparaciones aritméticas entre variables numéricas que no necesariamente deben ocurrir se cumplen para todas las llamadas del conjunto de pruebas. Estos falsos positivos aparecen principalmente cuando se comparan dos campos numéricos cuyos valores se encuentran en distintos órdenes de magnitud. Por ejemplo, uno de los invariantes detectados en la operación "Get Album tracks" de la API de Spotify establece que la duración en milisegundos de una canción debe ser mayor que su número de artistas (`return.duration_ms >size(return.artists[])`). Este tipo de invariantes son difíciles de descartar de manera automática, ya que puede resultar muy difícil encontrar un contraejemplo que los anule (si es que existe).

El diagrama de proyección solar de la Figura 5.1 muestra la distribución de los distintos tipos de falsos positivos detectados en los conjuntos de pruebas compuestos por 1000 llamadas. Puede observarse que los más comunes son los relacionados con comparaciones aritméticas que no necesariamente deben cumplirse (constituyendo un 57.7 % de los falsos positivos).

En vista de estos resultados, podemos responder a RQ1 de la siguiente forma:

RQ1: La propuesta es efectiva generando oráculos de prueba válidos a partir de un conjunto de casos de prueba, obteniendo una precisión total del 66.5 % y generando oráculos que modelan comportamientos no triviales del dominio de sistemas reales de gran complejidad.

5.3.2. Impacto del tamaño del conjunto de pruebas en la calidad de los oráculos generados

La gráfica de la Figura 5.2 muestra la evolución del valor de la precisión obtenida por la propuesta (Columna "P(%)" de la Tabla 5.2) a medida que incrementa el número de casos de prueba utilizados para la detección de invariantes. Puede observarse que no existe una variación significativa para 6 de las 8 operaciones utilizadas para la evaluación. Las únicas operaciones en las que se produce un cambio significativo de la precisión son la operación "Search businesses" de la API de Yelp y la operación "By ID or Title" de OMDb.

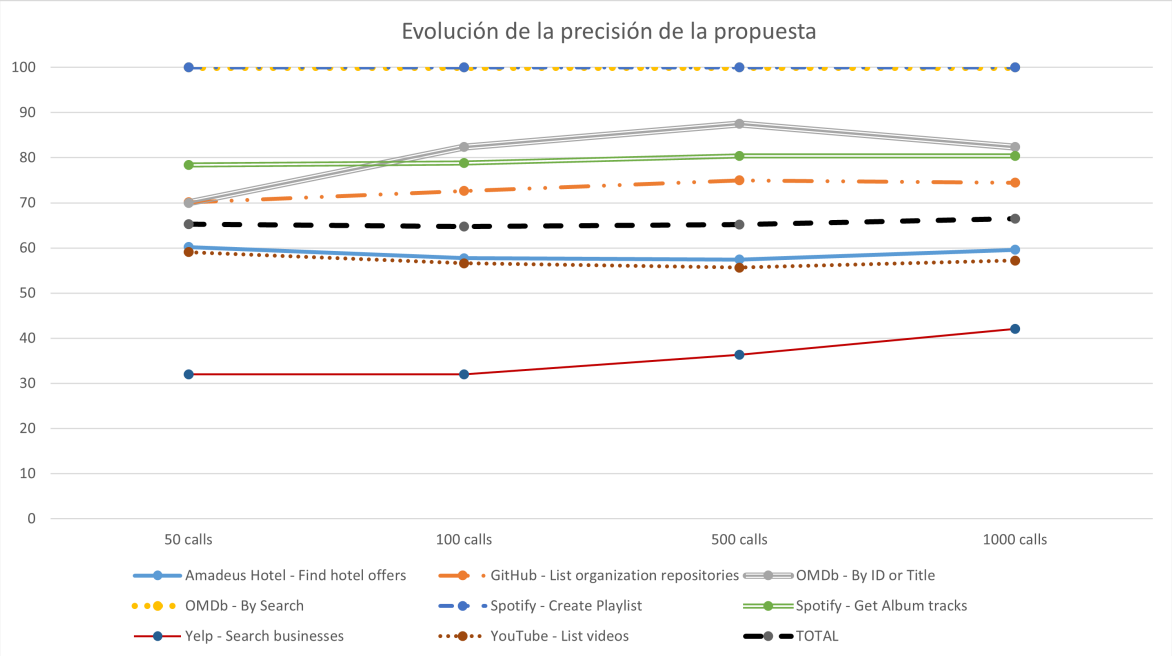

Figura 5.2: Evolución de la precisión de la propuesta

En el caso de la API de Yelp, el aumento del número de casos de prueba ha permitido la eliminación de invariantes que establecían comparaciones aritméticas que no necesariamente deben cumplirse, así como invariantes que especificaban que un valor de la respuesta solo tenía un valor concreto. Por ejemplo, en los 50 primeros casos de prueba se había cumplido que el número de reseñas de todos los establecimientos era mayor que el número de tipos de transacciones que soportaba (`return.review_count >size(return.transactions[])`). Además, para las primeras 500 llamadas se cumplió que todos los establecimientos estaban abiertos (`return.is_closed == false`).

Al generar invariantes potenciales a partir de un conjunto de 50 pruebas de la operación "By ID or Title" de la API de OMDb, la mayoría de los falsos positivos establecen que ciertos campos deben tener un conjunto de valores muy concreto (e.g., `return.Episode one of {"14", "8"}`). Todos estos falsos positivos, exceptuando uno, no están presentes al procesar el conjunto de 100 llamadas, y desaparecen por completo al procesar el conjunto de 500. La razón por la que se produce un decremento en la precisión al alcanzar las 1000 llamadas se debe a que existe una única llamada en

todo el conjunto de pruebas que devuelva un valor para el campo `Production` de la respuesta, lo que resulta en la aparición del invariante `return.Production == "Crappy World Films"`.

En vista de estos resultados, podemos responder a RQ2 como sigue:

> **RQ2:** El incremento del número de casos de prueba utilizados para la detección de invariantes no supone una variación significativa en la precisión obtenida. Solo se ha producido una variación apreciable en 2 de las 8 operaciones utilizadas para realizar la evaluación.

5.3.3. Capacidad de detección de errores de la propuesta

Los oráculos generados pueden implementarse como assertions que permitan detectar errores en futuras respuestas de la API. No obstante, si la API muestra un comportamiento erróneo, existe la posibilidad de que esto se vea reflejado en alguno de los invariantes detectados, lo que permite detectar bugs en la implementación o inconsistencias en la documentación al interpretarlos.

Las columnas 4, 7, 10 y 13 de la Tabla 5.2 muestran el número de invariantes por operación que revelan la existencia de un bug o una inconsistencia. En total, se han detectado 12 invariantes de este tipo en 3 de las 6 APIs usadas para la evaluación. Dichos invariantes han permitido identificar 6 bugs replicables en 4 de las 8 operaciones. Nótese que, exceptuando el caso de un invariante de este tipo en la API de Amadeus para cuya detección ha sido necesario emplear 100 casos de prueba, para detectar el resto de bugs solo se ha requerido de 50 casos de prueba. En los siguientes apartados se describen los errores detectados en cada API.

Amadeus Hotel – Habitaciones sin camas

La operación "Find hotel offers" de la API de Amadeus Hotel permite buscar ofertas de habitaciones de hoteles a partir de filtros como el id del hotel o la fecha de entrada y salida, entre otros. Una de las propiedades de la respuesta contiene las características de la habitación a reservar, entre las que se encuentra su número de camas. Según la especificación OAS oficial de la API (Extracto de código 5.1), el número de camas de una habitación debe oscilar entre 1 y 9.

```
HotelProduct_EstimatedRoomType:
    type: object
    properties:
        category:
        ...
        beds:
            description: 'Number of beds in the room (1-9)'
            type: integer
        bedType:
        ...
```

Extracto de código 5.1: Fragmento de la especificación OAS de Amadeus Hotel.

Sin embargo, uno de los invariantes detectados especifica que el número de camas de una habitación debe ser mayor o igual que 0 (`return.room.typeEstimated.beds >= 0`), lo que revela que algunas de las habitaciones devueltas por los casos de prueba contienen 0 camas. Al analizar el conjunto de respuestas, se han detectado un total de 55 ofertas de hoteles en las que la habitación tenía 0 camas. A pesar de esto, los datos de la habitación contenían información sobre el tipo de cama (como se muestra en el Extracto de código 5.2).

```
"room":{
    "type":"A0K",
    "typeEstimated":{
        "category":"SUITE",
        "beds":0,
        "bedType":"KING"
    }
}
```

Extracto de código 5.2: Respuesta con 0 camas.

GitHub – Inconsistencia en la documentación

La operación "List organization repositories" de la API de GitHub permite, dado el identificador de una organización de GitHub, listar todos sus repositorios. De acuerdo con la documentación oficial (tanto la documentación web como la especificación OAS), uno de los campos de cada repositorio devuelto es un objeto que contiene las propiedades del repositorio plantilla (conocido como template repository) usado para la creación de cada uno de los repositorios (en caso de que no se haya usado un repositorio plantilla, este campo será nulo).

A pesar de que algunos de los repositorios de las organizaciones usadas para la creación del conjunto de pruebas utilizaban repositorios plantillas, 9 de los invariantes detectados por la propuesta especifican que todas las propiedades del campo `template_repository` de la respuesta son nulos (e.g., `return.template_repository==null`).

Este invariante ha permitido detectar que la propiedad `template_repository` nunca está presente en la respuesta de la API. Este comportamiento ya ha sido notificado a los proveedores de la API, que han confirmado que se trata de una inconsistencia en la documentación y han creado una issue interna para actualizarla (Figura 5.3). El siguiente vídeo muestra la replicación de esta inconsistencia [11].

OMDb – Valor de parámetro no especificado en la documentación

El parámetro `type` de las operaciones "By ID or Title" y "By Search" de la API de OMDb puede utilizarse para filtrar los resultados devueltos por la API a un tipo concreto de producción, que puede ser "movie", "series" o "episode", tal y como se especifica en la documentación oficial (Figura 5.4).

Sin embargo, uno de los invariantes detectados en la operación "By Search" muestra que este filtrado por tipo no funciona de la forma especificada en la documentación.

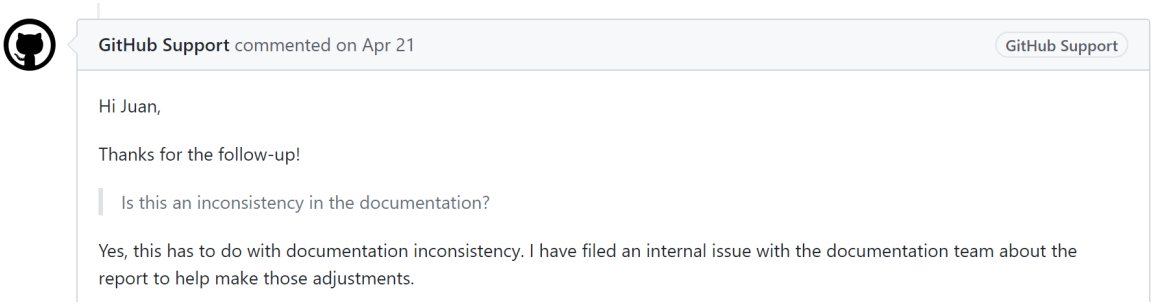

Figura 5.3: Confirmación del error por parte de los proveedores de la API de GitHub.

By Search

Parameter	Required	Valid options	Default Value	Description
s	Yes		<empty>	Movie title to search for.
type	No	movie, series, episode	<empty>	Type of result to return.

Figura 5.4: Parámetro type de la API de OMDb.

Dicho invariante es `return.Type one of {"game", "movie", "series"}`. Este invariante ha permitido identificar 2 bugs e inconsistencias en esta operación. La primera de estas inconsistencias es que es posible establecer el valor del parámetro `type` como "game", valor no especificado en la documentación (en el siguiente vídeo puede verse el uso de este valor para el parámetro `type` [15]).

El segundo bug consiste en que no es posible filtrar los resultados por episodio (de serlo, el invariante anterior incluiría "episode" como uno de los valores). Se ha detectado que tampoco es posible realizar este filtrado en la operación "By ID or Title", por lo que se han detectado un total de 3 bugs en esta API a partir de un solo invariante. La replicación de este bug en ambas operaciones puede observarse en el siguiente vídeo [14].

OMDb – Puntuaciones inconsistentes

El campo `imdbRating` de la respuesta de la operación "By ID or Title" de OMDb contiene la nota numérica, del 0 al 10, y con una sola cifra decimal, que ha recibido una producción. Uno de los invariantes detectados especifica que la longitud de este campo siempre es 3 (`LENGTH(return.imdbRating)==3`), lo que revela ninguna de las producciones tiene una puntuación de 10.0. Sin embargo, algunas de las predicciones del conjunto de datos sí tienen una puntuación de 10.0 en la página oficial de IMDb, pero la API devuelve un 9.9, lo que revela una inconsistencia en los datos de la API. El siguiente vídeo muestra la replicación de esta inconsistencia [16].

En vista de estos resultados, podemos responder a RQ3 de la siguiente forma:

RQ3: La propuesta es efectiva detectando errores a partir de un conjunto de casos de prueba. En total, se han identificado un total de 6 bugs replicables en 4 operaciones pertenecientes a 3 APIs comerciales con millones de usuarios.

6. Amenazas a la validez

En este capítulo se discuten las posibles amenazas a la validez, tanto internas como externas, que puedan haber influido en este trabajo, y como estas han sido mitigadas.

6.1. Validez interna

¿Existen factores que puedan haber afectado a los resultados de la evaluación? Para la experimentación, se ha utilizado la especificación OAS de cada una de las APIs que componen el conjunto de evaluación. Cuando ha sido posible, se han utilizado especificaciones que estuviesen disponibles públicamente. Sin embargo, en el caso de OMDb y Yelp, ha sido necesario crear esta especificación manualmente a partir de la documentación web. Por lo tanto, existe el riesgo de que alguna estas especificaciones contenga errores que las diferencie de la documentación web. Para mitigar esta posible amenaza, cada una de estas especificaciones ha sido revisada exhaustivamente. Además, RESTest comprueba automáticamente que no exista ninguna inconsistencia entre la especificación y la respuesta devuelta por la API para cada caso de prueba, algo que no ha ocurrido en ninguna de las pruebas que conforman nuestro conjunto de datos.

La variedad de las respuestas devueltas por las APIs depende en gran medida de los valores establecidos para cada uno de los parámetros de entrada, por lo que un conjunto de valores de entrada muy pequeño o poco variado podría resultar en sesgos. Para ejercitar la lógica de la API lo máximo posible, se ha buscado seleccionar un conjunto de valores de entrada amplio y variado para cada uno de los parámetros de entrada. Por ejemplo, en el caso de la API de Amadeus, se han aportado un total de 262 códigos de hoteles distintos, pertenecientes a distintos países y continentes.

La clasificación de cada uno de los invariantes detectados como verdaderos positivos o falsos positivos puede verse afectada por sesgos humanos o errores a la hora de interpretar el dominio de la API. Para mitigar esta amenaza, la documentación (tanto la especificación OAS como la versión web) de cada API ha sido revisada exhaustivamente antes de clasificar cada uno de los invariantes, contactando a los desarrolladores cuando fuese necesario aclarar posibles ambigüedades.

Por último, la existencia de errores en el instrumenter implementado podría comprometer la validez de los resultados. Para mitigar esto, el sistema ha sido probado de forma exhaustiva, con un total de 61 tests unitarios que abarcan toda la funcionalidad del sistema.

6.2. Validez externa

¿Hasta qué punto se pueden generalizar los resultados de esta investigación? La propuesta presentada en este trabajo ha sido evaluada con un subconjunto de 8 operaciones de 6 APIs distintas, por lo que las conclusiones obtenidas podrían no generalizarse más allá de estas operaciones. Para mitigar esta posible amenaza, se ha buscado evaluar la propuesta con un conjunto de APIs pertenecientes a dominios diferentes y con un número variado de operaciones y parámetros, que ya han sido utilizadas previamente en evaluaciones de otras propuestas para la generación automática de pruebas en APIs RESTful.

Por otra parte, se ha extendido Daikon añadiendo un total de 22 nuevos invariantes. El uso de estos tipos de invariantes ha permitido generar una gran cantidad de oráculos en el conjunto de APIs usados para la evaluación, pero existe la posibilidad de que estos resultados no se puedan generalizar. Sin embargo, cabe destacar que estos invariantes han sido generados a partir de un análisis del comportamiento de un dataset de 48 APIs creado por el autor en una publicación anterior [34]. Igualmente, es importante señalar que el conjunto de tipos de invariantes no pretende ser completo y que cabe la posibilidad de añadir nuevos tipos de invariantes en el futuro.

7. Conclusiones y trabajo futuro

En este trabajo se ha presentado una propuesta para la generación automática de oráculos de prueba para APIs RESTful. Dichos oráculos se obtienen mediante la instrumentalización de la especificación OAS de la API y de un conjunto de casos de prueba para los que se conocen únicamente sus entradas y salidas. Al no requerir de acceso al código fuente de la API, la propuesta opera en un contexto de caja negra, lo que la hace aplicable a cualquier API RESTful, independientemente de la tecnología utilizada para su implementación.

La evaluación realizada sobre un conjunto de 8 operaciones de 6 APIs comerciales muestra la capacidad de la propuesta para generar oráculos válidos a partir de un conjunto de casos de prueba, llegando a obtener una precisión del 100 % en algunos casos. Además, la aplicación de la propuesta ha permitido detectar un total de 6 bugs replicables en 3 APIs comerciales con millones de usuarios (Amadeus, GitHub y OMDb).

A continuación, se presentan las principales líneas de trabajo futuro identificadas:

- **Generación automática de assertions:** Actualmente, los invariantes devueltos por la propuesta se especifican en un formato que, aunque es fácil de interpretar por un humano, no puede utilizarse directamente para generar assertions que puedan insertarse directamente en un código escrito en un lenguaje de programación específico. Ahora que se ha comprobado la viabilidad de la propuesta para generar oráculos de prueba y detectar errores, la generación automática de estas assertions es el primer paso a realizar en la extensión de este trabajo.
- **Despliegue como API:** Esta mejora permitiría que la propuesta fuese accesible para cualquier usuario con acceso a internet, omitiendo la necesidad de instalar y configurar todas las herramientas necesarias para su uso.
- **Extracción y validación de nuevos valores para los parámetros de entrada:** El autor de esta propuesta ha diseñado e implementado un método para la generación automática de valores de entrada a partir de la especificación OAS de una API, que ya ha sido publicado [34]. Los invariantes detectados podrían usarse para mejorar dicha herramienta mediante la validación automática de los valores generados (por ejemplo, detectando que el uso de un valor siempre resulta en un mensaje de error) o para obtener nuevos valores de entrada a partir de las respuestas de la API. Esto último podría lograrse explotando la información aportada por aquellos invariantes que revelan que el valor de un parámetro de la entrada siempre es igual que un campo de la salida.

A. Bibliografía

[1] APIFuzzer website. https://github.com/KissPeter/APIFuzzer. Accessed June 2022.

[2] Evomaster website. https://github.com/EMResearch/EvoMaster. Accessed June 2022.

[3] OMDb API. http://www.omdbapi.com. Accessed May 2022.

[4] Amadeus Hotel Shopping API. https://developers.amadeus.com/self-service/category/hotel/api-doc/hotel-search/api-reference. Accessed May 2022.

[5] bBoxrt website. https://git.dei.uc.pt/cnl/bBOXRT. Accessed June 2022.

[6] BeSoccer API. https://www.besoccer.com/api/. Accessed May 2022.

[7] Daikon developer manual. https://plse.cs.washington.edu/daikon/download/doc/developer.html, . Accessed May 2022.

[8] Daikon user manual. https://plse.cs.washington.edu/daikon/download/doc/daikon.html, . Accessed June 2022.

[9] DeepL API. https://www.deepl.com/docs-api/. Accessed May 2022.

[10] Dredd website. https://github.com/apiaryio/dredd. Accessed June 2022.

[11] Bug GitHub API - Repository template. https://www.youtube.com/watch?v=sdA9Gc4usSg&ab_channel=JuanCarlosAlonsoValenzuela, . Accessed June 2022.

[12] GitHub API - List organization repositories. https://docs.github.com/en/rest/repos/repos, . Accessed June 2022.

[13] HTTP status codes. https://www.restapitutorial.com/httpstatuscodes.html. Accessed June 2022.

[14] OMDb API - Cannot filter by episode. https://www.youtube.com/watch?v=5cXIHDDwpNE&ab_channel=JuanCarlosAlonsoValenzuela, . Accessed June 2022.

[15] OMDb API - type game. https://www.youtube.com/watch?v=p_NMY4bBGFc&ab_channel=JuanCarlosAlonsoValenzuela, . Accessed June 2022.

[16] OMBb API - Inconsistent IMDbRating. https://www.youtube.com/watch?v=K6MEHhZOJGs&ab_channel=JuanCarlosAlonsoValenzuela, . Accessed June 2022.

[17] OpenAPI Specification. https://www.openapis.org, . Accessed May 2022.

[18] Openroute API. https://openrouteservice.org/dev/#/api-docs, . Accessed May 2022.

[19] Programmable Web API directory. https://www.programmableweb.com. Accessed May 2022.

[20] RapidAPI API directory. https://rapidapi.com/marketplace. Accessed May 2022.

[21] Replication package. https://doi.org/10.5281/zenodo.6687916. Accessed June 2022.

[22] RESTest website. https://github.com/isa-group/RESTest, . Accessed June 2022.

[23] Restler website. https://github.com/microsoft/restler-fuzzer, . Accessed June 2022.

[24] RestTestGen website. https://github.com/SeUniVr/RestTestGen, . Accessed June 2022.

[25] SchemaThesis website. https://github.com/schemathesis/schemathesis. Accessed June 2022.

[26] Spotify API - Create Playlist. https://developer.spotify.com/console/post-playlists/, . Accessed June 2022.

[27] Spotify API - Get Album Tracks. https://developer.spotify.com/console/get-album-tracks/, . Accessed June 2022.

[28] Dredd website. https://github.com/Cornutum/tcases/tree/master/tcases-openapi. Accessed June 2022.

[29] Twitter API. https://developer.twitter.com/en/docs. Accessed May 2022.

[30] Yelp Fusion API - Businesses Search. https://www.yelp.com/developers/documentation/v3/business_search. Accessed June 2022.

[31] YouTube API - List Videos. https://developers.google.com/youtube/v3/docs/videos/list. Accessed June 2022.

[32] DAIKON front-ends. https://plse.cs.washington.edu/daikon/download/doc/daikon.html#Front-ends-_0028instrumentation_0029, 2022. Accessed June 2022.

[33] Afsoon Afzal, Claire Le Goues, and Christopher Steven Timperley. Mithra: Anomaly detection as an oracle for cyberphysical systems. *IEEE Transactions on Software Engineering*, pages 1–1, 2021. doi: 10.1109/TSE.2021.3120680.

[34] Juan C. Alonso, Alberto Martin-Lopez, Sergio Segura, Jose Maria Garcia, and Antonio Ruiz-Cortes. ARTE: Automated Generation of Realistic Test Inputs for Web APIs. *IEEE Transactions on Software Engineering*, 2022. doi: 10.1109/TSE.2022.3150618.

[35] Juan C. Alonso, Sergio Segura, and Antonio Ruiz-Cortés. Generación Automática de Oráculos de Prueba para APIs RESTful. In *Jornadas de la Sociedad de Ingeniería de Software y Tecnologías de Desarrollo de Software (SISTEDES)*. SISTEDES, 2022.

[36] apisguru. APIs.guru. https://apis.guru. Accessed June 2022.

[37] Andrea Arcuri. Restful api automated test case generation with evomaster. *ACM Transactions on Software Engineering and Methodology*, 28:1–37, 2019. doi: 10.1145/3293455.

[38] V. Atlidakis, P. Godefroid, and M. Polishchuk. Restler: Stateful rest api fuzzing. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 748–758, 2019. doi: 10.1109/ICSE.2019.00083.

[39] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2014.

[40] Jake Cobb, James A. Jones, Gregory M. Kapfhammer, and Mary Jean Harrold. Dynamic invariant detection for relational databases. In *Proceedings of the Ninth International Workshop on Dynamic Analysis*, WODA '11, page 12–17, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450308113. doi: 10.1145/2002951.2002955. URL https://doi.org/10.1145/2002951.2002955.

[41] Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, 1992.

[42] Michael D Ernst, Jake Cockrell, William G Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE transactions on software engineering*, 27(2):99–123, 2001.

[43] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1):35–45, 2007. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2007.01.015. URL https://www.sciencedirect.com/science/article/pii/S016764230700161X. Special issue on Experimental Software and Toolkits.

[44] Michael Dean Ernst. *Dynamically discovering likely program invariants*. University of Washington, 2000.

[45] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Irvine, 2000.

[46] Andrea Fioraldi, Daniele Cono D'Elia, and Davide Balzarotti. The use of likely invariants as feedback for fuzzers. 08 2021.

[47] Cristiano Giuffrida, Lorenzo Cavallaro, and Andrew S Tanenbaum. Practical automated vulnerability monitoring using program state invariants. In *2013 43rd*

Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 1–12. IEEE, 2013.

[48] Sudheendra Hangal and Monica S Lam. Tracking down software bugs using automatic anomaly detection. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 291–301. IEEE, 2002.

[49] Zac Hatfield-Dodds and Dmitry Dygalo. Deriving semantics-aware fuzzers from web api schemas. *arXiv preprint arXiv:2112.10328*, 2021.

[50] Deborah S Katz, Christopher S Timperley, and Claire Le Goues. Using dynamic binary instrumentation to detect failures in robotics software. *arXiv preprint arXiv:2201.12464*, 2022.

[51] Myeongsoo Kim, Qi Xin, Saurabh Sinha, and Alessandro Orso. Automated test generation for rest apis: No time to rest yet. *arXiv preprint arXiv:2204.08348*, 2022.

[52] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs. In *International Conference on Service-Oriented Computing*, pages 459–475, 2020.

[53] Ali Mesbah, Arie van Deursen, and Danny Roest. Invariant-based automatic testing of modern web applications. *IEEE Transactions on Software Engineering*, 38(1):35–53, 2012. doi: 10.1109/TSE.2011.28.

[54] Carlos Pacheco and Michael D. Ernst. Eclat: Automatic generation and classification of test inputs. In Andrew P. Black, editor, *ECOOP 2005 - Object-Oriented Programming*, pages 504–527, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31725-8.

[55] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[56] Leonard Richardson, Mike Amundsen, and Sam Ruby. *RESTful Web APIs*. O'Reilly Media, Inc., 2013. ISBN 1449358063, 9781449358068.

[57] David Schuler, Valentin Dallmeier, and Andreas Zeller. Efficient mutation testing by checking invariant violations. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, page 69–80, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605583389. doi: 10.1145/1572272.1572282. URL https://doi.org/10.1145/1572272.1572282.

[58] Sergio Segura, José A. Parejo, Javier Troya, and Antonio Ruiz-Cortés. Metamorphic testing of restful web apis. *IEEE Transactions on Software Engineering*, 44(11):1083–1099, 2018. doi: 10.1109/TSE.2017.2764464.

[59] Dimitri Stallenberg, Mitchell Olsthoorn, and Annibale Panichella. Improving test case generation for rest apis through hierarchical clustering. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 117–128, 2021. doi: 10.1109/ASE51524.2021.9678586.

[60] E. Viglianisi, M. Dallago, and M. Ceccato. Resttestgen: Automated black-box testing of restful apis. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 142–152, 2020. doi: 10.1109/ICST46399.2020.00024.

[61] Xiaodong Yang, Omar Ali Beg, Matthew Kenigsberg, and Taylor T. Johnson. A framework for identification and validation of affine hybrid automata from input-output traces. *ACM Trans. Cyber-Phys. Syst.*, 6(2), apr 2022. ISSN 2378-962X. doi: 10.1145/3470455. URL https://doi.org/10.1145/3470455.

[62] Yuntong Zhang, Xiang Gao, Gregory J Duck, and Abhik Roychoudhury. Program vulnerability repair via inductive inference. 2022.