Metamorphic Relation Patterns for Query-Based Systems

Sergio Segura, Amador Durán, Javier Troya, and Antonio Ruiz-Cortés
Department of Computer Languages and Systems
Universidad de Sevilla
Spain

***Abstract*—Searching and displaying data based on user queries is a key feature of most software applications such as information systems, web portals, web APIs, and data analytic platforms. The large volume of data managed by these types of systems, henceforth called query-based systems (QBS), makes them extremely hard to test due to the difficulty to assess whether the output of a query is correct, the so-called *oracle problem*. Metamorphic testing has proved to be a very effective approach to alleviate the oracle problem in QBS, enabling the detection of bugs in data repositories, large e-commerce sites, and some of the most used software applications worldwide such as Google Search and YouTube. We have observed, however, that the metamorphic relations used to test different types of QBS are very similar, regardless of their domain, since all of them exploit standard query features such as filtering and ordering. Inspired by this finding, in this paper we present a catalogue of metamorphic relation patterns to assist testers in the identification and inference of metamorphic relations in QBS. For the definition of the patterns we resorted to the root of most query languages: relational algebra. We show how the proposed patterns help in the identification of metamorphic relations in the e-commerce platform PrestaShop, the email service Gmail, and the mobile application of video streaming HBO.**

***Index Terms*—Metamorphic testing, metamorphic relation, metamorphic relation pattern, query-based system.**

I. Introduction

Most software applications support searching and displaying data based on user queries. A *query* specifies user preferences—the data to be retrieved and how to display it—declaratively using visual forms or textual languages, which typically support standard operations such as filtering, ordering or pagination. In what follows, we refer to software systems supporting user queries as *query-based systems* (QBS). Typical examples of QBS are information systems like OpenBravo (supporting queries like "retrieve invoices issued before 2019"), e-commerce sites like Amazon (e.g., "search for cameras under $100"), software project platforms like GitHub (e.g., "get Node.js projects with less than 10 committers"), video streaming apps like HBO (e.g., "order series by popularity"), email clients like Gmail (e.g., "display messages including the word *SEWORLD* in the subject"), or even video games like World of Warcraft (e.g., "get progression data for character *Thrall* in *Blackrock* realm"). Testing QBS is extremely challenging as they suffer from the oracle problem: it is very difficult, often infeasible, to assess whether the output of a query is correct, either because the expected output is unknown or because it is hard to compare it to the observed output [1], [2].

Metamorphic testing [3]–[6] has been successfully applied to alleviate the oracle problem in different types of QBS including search engines such as Google and Bing [7]–[9], RESTful APIs such as Spotify and YouTube [10], e-commerce sites such as Amazon and Walmart [11], and data repositories like the NASA's Data Access Toolkit [12]. Metamorphic relations in QBS specify the relation between the inputs and outputs of multiple queries. For example, suppose a search for videos in YouTube using the keyword "ICSE" that returns 1245 results. Checking whether this output is correct would be hard and time-consuming, even having access to the internal systems of YouTube. Suppose that a new search is performed with the same keyword restricting the search to videos in high definition (HD), returning 710 results. As expected, the number of HD videos of ICSE is lower than the total number of videos of ICSE. Otherwise, the comparison of both queries would have revealed a failure. This is an example of a simple metamorphic relation: the number of videos returned in a search for the keyword K (source test case) should be equal or higher than the number of videos returned in a search for K using the HD filter (follow-up test case).

It is worth observing that most QBS support standard operations such as filtering, ordering and pagination, which makes them exhibit the same types of metamorphic relations. For example, the idea behind the previous metamorphic relation could be potentially applicable to any system supporting queries with filters, regardless of how these are specified. In Amazon, for instance, we could run a search query for "watches" (source test case) and then repeat the search using a maximum price filter of $100 (follow-up test case). In the Netflix mobile app, we could search for "films" (source test case) and then repeat the search setting the genre filter to "comedy" (follow-up test case). In IEEE Xplore, we could search for papers related to "metamorphic"(source test case) and then restrict the search adding a new keyword as "metamorphic AND testing" (follow-up test case). In all cases, the number of results of the first search should be equal or higher than the number of results of the second search (filtered). These relations reflect a *pattern* applicable to most QBS: the number of results of a query (source test case) should be equal or higher than the number of results of the same query when adding one or more filters (follow-up test case). Inspired by

978-1-7281-2235-9/19/$31.00 ©2019 IEEE
DOI 10.1109/MET.2019.00012

this finding, the concept of metamorphic relation "patterns" has been proposed as abstract relations from which multiple concrete metamorphic relations can be instantiated [10], [11], [13]. Zhou et al. [11] defines a *metamorphic relation pattern* (MRP) as an abstraction that characterizes a set of (possibly infinitely many) metamorphic relations. The previous pattern, for instance, is an example of MRP from which we could instantiate numerous metamorphic relations in multiple systems under test by using different types of queries, different filters, and even different number of follow-up test cases.

The use of patterns has two main benefits. First, they can be extremely helpful for identifying metamorphic relations [10], [11], [13]. This is because patterns guide testers on the search for metamorphic relations with a certain structure, making the identification of the relations significantly easier than when starting from scratch. Second, the identification of patterns is a key point to enable the automated inference of likely metamorphic relations that conform to those patterns [14]. For example, papers on the automated inference of likely metamorphic relations in numerical and machine learning programs (e.g., [15], [16]) typically search for instances of the "metamorphic properties" (similar to the concept of pattern) defined by Murphy et al. back in 2008 [17]. For example, the *permutative* property specifies that the order of the inputs should not affect the output (c.f. Section II-A).

In this paper, we present a catalogue of seven MRPs to assist testers in the identification and inference of metamorphic relations in QBS. The proposed patterns are based on previous work of the authors and also on common metamorphic relations found in related papers on metamorphic testing of QBS. For the definition of the patterns, we use relational algebra [18], typically used to define the semantics of query languages like SQL. To illustrate the feasibility of the approach, we show how the proposed patterns help in the identification of metamorphic relations in three real QBS with millions of users: the e-commerce platform PrestaShop [19], the web email service Gmail [20], and the video streaming app of HBO [21].

The reminder of the paper is structured as follows. Section II introduces MRPs and relational algebra. The catalogue of patterns is presented in Section III. Section IV describes some of the metamorphic relations that can be derived from the patterns in three different types of QBS. Finally, we summarize our conclusions and potential future work in Section V.

II. Background

A. Metamorphic relation patterns

In previous work the authors introduced the term "metamorphic relation output pattern" (MROP) as an abstract relation among the source and follow-up outputs from which multiple concrete metamorphic relation can be derived [10]. This work opened a new MT research direction on "metamorphic relation patterns", in a broad sense, as foreseen by Segura in his keynote as the Third International Workshop on Metamorphic Testing (ICSE MET'18) [13]. Zhou et al. [10] further investigated the notion of "patterns" and formally presented the general concept of "metamorphic relation pattern (MRP)" as an abstraction that characterizes a set of (possibly infinitely many) metamorphic relations. MRPs ease the systematic identification of metamorphic relations and can serve as a starting point for the automated inference of metamorphic relations that conform to the patterns. Although the concept of patterns has been recently introduced, the intuitive idea behind patterns had been explored earlier. Zhou et al. [7] used the term *general metamorphic relations*, analogous to the concept of MRP, in their paper on testing search engines back in 2007. Short after that, Murphy et al. [17] introduced the concept of *metamorphic property*, also aligned to the concept of pattern, as a general form of metamorphic relation that "*provide a foundation for determining the relationships and transformations that can be used for conducting metamorphic testing*" in machine learning applications. More specifically, they identified six properties (patterns) called additive, multiplicative, permutative, invertive, inclusive, and exclusive. As an example, the *additive* property specifies that modifying the input data by addition should not affect the output. These properties are possibly the most successful examples of patterns so far having been used by different research groups for the identification and inference of metamorphic relations in several machine learning and numerical programs, e.g., [15], [16], [22]–[24]. Troya et al. [14] identified a set of *domain-independent trace patterns* to automatically infer likely metamorphic relations in the context of model transformations.

An MRP can also be specified as an incomplete metamorphic relation where only the relation among the inputs or the outputs is specified. This has motivated the introduction of two subclasses of patterns: *metamorphic relation input pattern* (MRIP) [11] and *metamorphic relation output pattern* (MROP) [10]. An MRIP is defined as an abstraction that characterizes the relations among the source and follow-up inputs of a set of metamorphic relations [11]. Conversely, an MROP describes an abstract relation among the source and follow-up outputs, but not the relation among the inputs.

In a previous paper, we proposed six MROPs for the identification of metamorphic relations in RESTful web services [10]: equivalence, equality, subset, disjoint, complete and difference. The *disjoint* pattern, for example, represents those metamorphic relations where the intersection among the source and follow-up outputs should be empty, although it does not specify how the source test case should be transformed into a follow-up test case such that the output relation holds. The patterns presented in this paper are highly inspired on the MROPs presented in our previous work, based on the observation that they could be applicable to most QBS, and not just web services. However, in this paper we go a step further by specifying complete patterns—MRPs rather than MROPs—where both the input and the output relations are specified.

In a recent paper, Zhou et al. [11] proposed a *symmetry* MRP and a *change direction* MRIP, based on the observation that most systems can be observed from different viewpoints from which the system appear the same. For example, an AI-

enabled object recognition system should recognize the same objects in a video, regardless of whether it is played forwards or backwards.

B. Relational algebra

Relational algebra is a theoretical query language proposed by the creator of the relational model E. F. Codd [25]. Apart from set–like operators like Cartesian product ($\times$), union ($\cup$), difference ($-$) or intersection ($\cap$), relational algebra also includes projection (Π), selection (σ), division ($\div$) and join ($\bowtie$) operators [18]. Some extensions to relational algebra have been proposed, including the τ operator, that sorts a relation on the values of a list of attributes generating a *sequence* [18].

In a nutshell, the relational algebra operators that are mainly used in the rest of the paper are listed below, assuming that a relation R exists and can be queried, e.g., products in Amazon. Notice that, instead of the usual subscript notation, we use parameters within parenthesis for the sake of readability.

- $\sigma(R, c)$ It returns the relation formed by all the tuples in R for which condition c holds. Notice that c is a *well-formed formula*, i.e., a simple condition or a compound condition using the $\wedge, \vee$ and $\neg$ logical connectors.
- $\tau(R, \langle a_i \rangle)$ It returns a sequence of all the tuples in R ordered by the value of the attributes in the sequence $\langle a_i \rangle_{i=1..n}$, i.e., first using the values of a_1, then those of a_2, etc. All attributes in $\langle a_i \rangle$ must be present in the tuples in R. If the sequence of attributes is empty, i.e. $\tau(R, \langle\rangle)$, the tuples are sorted using *default* ordering, which depends on the QBS being queried.

In the rest of the paper, we use the standard Z formal specification language [26], [27] for formalising concepts in the metamorphic relation patterns when needed, applying the previously mentioned relational algebra operators.

III. Metamorphic relation patterns

In this section, we present a catalogue of seven MRPs for QBS. These patterns are an abstraction of common (general) metamorphic relations found in related papers on metamorphic testing of QBS [7]–[10], [12]. These patterns are also highly inspired by the MROPs presented by the authors in the context of web APIs [10], based on the observation that they are also applicable to other types of QBS. However, in this paper we refine those patterns by considering not only the output relation among source and follow-up test cases, but also the relation among the inputs, i.e., we present MRPs rather than MROPs. This makes it easier to instantiate each pattern, although it also reduces their scope. For a more general view of the proposed patterns, we refer to the MROPs presented in our previous work [10]. Notice that we do not intend this to be a complete list of MRPs. In fact, it would be straightforward to identify new patterns as variations of the ones presented here or, for example, exploiting equivalences in relational algebra. In what follows, we briefly present each MRP including an example, references to related work, and its formalization using relational algebra.

A. Input equivalence

This pattern represents those relations where the source and the follow-up test cases are equivalent and therefore their outputs should contain the same items in the same order, i.e., they should be equal. There exist two typical situations where this pattern can be instantiated. The first one appears when input parameters accept equivalent values expressed in different formats, e.g., 1MB $\equiv$ 1024KB. The second situation appears in most queries and it is related to default values: it should not matter whether or not default values of the query parameters are specified. For example, a search in YouTube specifying no order should produce exactly the same result as indicating the default ordering criterion, which is based on the *relevance* to the search query. A key advantage of the relations instantiated from this pattern is that the outputs of source and follow-up test cases are easy to be compared since they are expected to be equal, e.g., using a diff tool. Lindvall et al. [12] used a rich set of these types of metamorphic relations to reveal failures in the NASA DAT, a large database of telemetry data. This pattern is a particular case of the *equality* MROP defined by the authors in [10].

Formally, let $q_1 = \tau(\sigma(R, c_1), \langle o_1 \rangle)$ and $q_2 = \tau(\sigma(R, c_2), \langle o_2 \rangle)$ be two queries defined on the same relation R. This MRP states that the outputs of q_1 and q_2 should be the same when conditions c_1 and c_2 and sequences of ordering criteria o_1 and o_2 are *equivalent*, i.e.,

$$(c_1 \equiv c_2 \ \wedge \ o_1 \equiv o_2) \Leftrightarrow (\tau(\sigma(R, c_1), \langle o_1 \rangle) = \tau(\sigma(R, c_2), \langle o_2 \rangle))$$

B. Shuffling

This pattern represents those metamorphic relations where the source and follow–up outputs should contain the same items regardless of the ordering criteria specified as input. For example, a search for "*hotels in London*" in Booking.com [28] should return the same results regardless of the ordering criteria specified (price, review score, starts, etc.). This pattern is a particular case of the *equivalence* MROP presented by the authors in the context of web APIs [10], and of the *change direction* MRIP recently proposed by Zhou et al. [11].

Formally, let $q = \sigma(R, c)$ be a query such that its result contains several attributes $\{a_i\}_{i=1..n}$ that can be used as ordering criteria with the τ operator defined in Section II-B. This MRP states that the result of the query q ordered by a given attribute a_i should contain the same elements than the same query ordered by any other different attribute a_j, i.e.,

$$\forall i, j : 1..n \mid i \neq j \bullet \ items\ \tau(q, \langle a_i \rangle) = items\ \tau(q, \langle a_j \rangle)$$

where *items* is a function that returns the bag of elements of a sequence (see [26, p. 127]), i.e., the unordered elements in the

query. Notice that this pattern can be generalized to consider any pair of sequences of the ordering attributes in $\{ a_i \}_{i=1..n}$

C. Conjunctive conditions

This pattern groups those relations where the query is iteratively refined adding new conjunctive conditions such that the results of each test case should be included in the results of the previous ones. This pattern is very common in query operations where most of the parameters are filters. For example, suppose we perform a search for YouTube videos of "pets". Next, we search for videos of "pets" in three dimension (3D), and finally we search for videos of "pets" in 3D uploaded after 2018. Intuitively, the results of the third search (videos of pets in 3D published after 2018) should be a subset of the result set of the second search (videos of "pets" in 3D), and in turn these should be a subset of the results of the original search (videos of "pets"). This pattern, and slight variations of it, were extensively used by Zhou et al. in their papers on testing search engines [7], [8], and also by Segura et al. in their paper on testing RESTful web APIs [10] (where it was generalized as the *subset* MROP).

Formally, let c be a complex condition formed by the conjunction of simpler conditions, i.e., $c = \bigwedge_{i=1..n} c_i$. This MRP states that the result of a query when one condition is added conjunctively to its selection condition should be a subset of the result of the query before adding the new condition, i.e.,

$$\forall\, i = 2\,..\,n \;\bullet\; \sigma(R, \bigwedge_{k=1..i} c_k) \;\subseteq\; \sigma(R, \bigwedge_{k=1..i-1} c_k)$$

D. Disjunctive conditions

This pattern is similar to the previous one, but the query is expanded with input disjunctive conditions such that the results of each test case should be a subset of the following ones. For example, let us suppose a search in IEEE Xplore for papers including the word "metamorphic" in their title. Next, we expand the search to papers including either "metamorphic" OR "testing". Naturally, the papers returned in the former search should be a subset of those found in the second search. This pattern has been previously exploited in the context of search engines [7], [8].

Formally, let c be a complex condition formed by the disjunction of simpler conditions, i.e., $c = \bigvee_{i=1..n} c_i$. This MRP states that the result of a query when one condition is added disjunctively to its selection condition should contain the result of the query before adding the new condition, i.e.,

$$\forall\, i = 2\,..\,n \;\bullet\; \sigma(R, \bigvee_{k=1..i-1} c_k) \subseteq\; \sigma(R, \bigvee_{k=1..i} c_k)$$

E. Disjoint partitions

This pattern represents those relations where the outputs of the follow–up test cases should be pairwise disjoint (i.e., they should have no items in common) because the underlying relation can be partitioned according to the values of at least one input attribute. For instance, suppose a search in a PayPal user's account [29] for refunds with status *COMPLETED*. Next, let us suppose a new search is performed in the same account for refunds with status *CANCELLED*. The results set of both searches should have no items in common. This pattern is a particular case of the *disjoint* MROP presented by the authors in the context of web APIs [10].

Formally, let $q = \sigma(R, c)$ be a query such that its result contains at least one attribute a_p whose domain is a discrete set of values, e.g., $\{v_1, v_2, \ldots v_n\}$. This MRP states that the result of two queries where a conjunctive condition of the form $a_p = v_i$ appears, should be disjoint if the values with which a_p is compared are different, i.e.,

$$\forall\, i,j : 1\,..\,n \;\mid\; i \neq j \;\bullet\; \sigma(R, c \wedge a_p{=}v_i) \;\cap\; \sigma(R, c \wedge a_p{=}v_j) \;=\; \varnothing$$

F. Complete partitions

This pattern is related to the previous one, and it represents those relations where the union of the follow–up outputs should contain the same items as the source output because the underlying relation can be partitioned according to the values of at least one input attribute. For instance, YouTube videos are classified according to its duration in *short* (less than 4 minutes), *medium* (between 4 and 20 minutes) and *long* videos (longer than 20 minutes). Consider a source test case consisting in a search for YouTube videos with the keyword "testing". Suppose that three follow–up test cases are constructed by searching for the same keyword restricting the search to short, medium, and long videos, respectively. Intuitively, the union of the follow–up test outputs (short, medium, and long videos) should contain the same videos as the source test output, where no duration filter was specified. This pattern is a particular case of the *complete* MROP presented by the authors in [10].

Formally, let $q = \sigma(R, c)$ be a query such that its result contains at least one attribute a_p whose domain is a discrete set of values, e.g., $\{v_1, v_2, \ldots v_n\}$. This MRP states that the union of the results of the n queries formed by adding a conjunctive condition of the form $a_p = v_i$ to the condition in q for each possible value of a_p, should be equal to the result of q, i.e.,

$$\sigma(R, c) \;=\; \bigcup_{i=1..n} \sigma(R, c \wedge a_p{=}v_i)$$

G. Partition difference

This pattern is derived from the two previous ones and represents those relations where the outputs of the follow–up test cases are pairwise disjoint and their union contains the same items as the source output because the queried relation can be partitioned according to the values of at least one input attribute. For instance, in the previous example the difference between all YouTube videos of "testing" and *long* videos about the topic should be equal to the union of *medium* and *short* videos. To the best of our knowledge, these types of metamorphic relations have not been exploited in the context of queries so far.

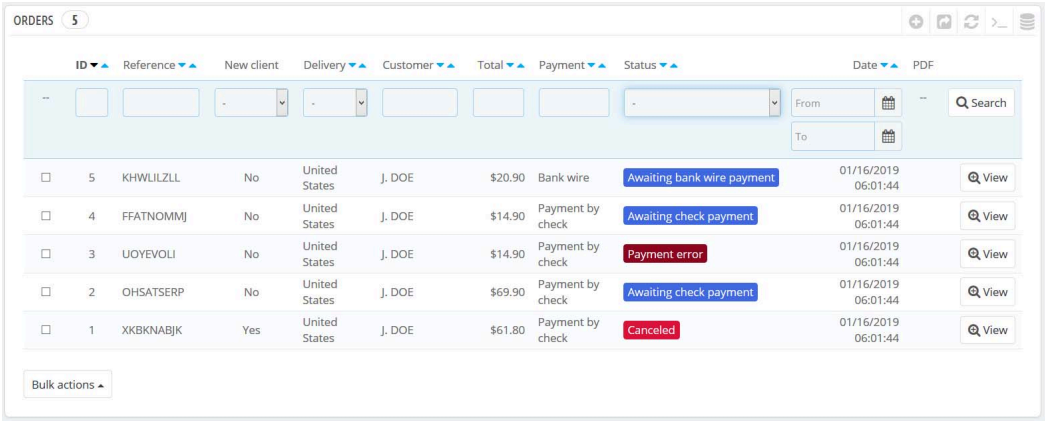

Figure 1. Order view in PrestaShop's back-office

Formally, let $q = \sigma(R, c)$ be a query such that its result contains at least one attribute a_p whose domain is a discrete set of values, e.g., $\{v_1, v_2, \ldots v_n\}$. This MRP states that the difference between q and the union of the results of the k queries formed by adding a conjunctive condition of the form $a_p = v_i$ to the condition in q for k different possible values of a_p, should be equal to the result of the union of the results of the $n-k$ queries formed by adding a conjunctive condition of the form $a_p = v_j$ to the condition in q for the $n-k$ different possible values of a_p, i.e.,

$$\forall k : 1\,..\,(n-1) \bullet \sigma(R,c) - \bigcup_{i=1..k} \sigma(R,\, c \wedge a_p = v_i) = \bigcup_{j=k+1..n} \sigma(R,\, c \wedge a_p = v_j)$$

IV. Identification of Metamorphic Relations

In this section, we show how the proposed MRPs can help in the identification of metamorphic relations in different types of QBS. To foster diversity, we excluded web search engines and web APIs since those types of QBS have been studied in detail in related papers [7]–[10]. In particular, we identified metamorphic relations in the e-commerce platform PrestaShop [19], the web email client Gmail [20] and the mobile application of video streaming HBO [21]. For each program, we next show some of the metamorphic relations identified indicating the MRP that they instantiate. We may remark that we do not intend to collect a complete list of metamorphic relations, but just a sample of the potentially huge number of relations that could be derived from the patterns on each program. We may also remark that the goal of this paper is not to study the fault detection capability of the identified metamorphic relations, but to study the validity and generalizability of the proposed MRPs. Notice that for each metamorphic relation identified, many metamorphic tests could be implemented by using specific test inputs and test fixtures, e.g., populating each QBS with different datasets.

Regarding the description of the relations, when not explicitly mentioned, the first query is the source test case and the subsequent queries are the follow-up test cases.

A. PrestaShop

PrestaShop [19] is an open source e-commerce platform written in PHP for the development of online shopping systems. It offers over 3,500 modules and visual templates powering more than 270,000 online stores worldwide. A PrestaShop store provides a public interface, where customers browse the site and place orders (front-office), and a private interface where administrators can manage the store (back-office). Figure 1 shows a screenshot of the orders view in the back-office of the online demo of PrestaShop v1.7.4[1]. A visual form is provided to browse and query orders by identifier, country, customer's name, amount, type of payment, status, and date. Next, we show some of the metamorphic relations that can be derived from the proposed patterns.

- MR_1 (Input equivalence). List the orders using default values. Then, get a new list ordered by `ID`, default ordering criterion. Both queries should return exactly the same result set.
- MR_2 (Shuffling). List the orders ordered by `Date`. Then, get a new list ordered by `Total` amount. Both queries should return the same orders, regardless of their ordering.
- MR_3 (Conjunctive conditions). List the orders placed after a date. Then, repeat the query adding a new filter to list orders paid by `Bank wire` only. The result set of the second query (follow-up test case) should be a subset of the result set of the first query (source test case), where no payment filter was applied.
- MR_4 (Disjoint partitions). List the orders with status `Delivered`. Then, repeat the query three more times changing the status filter to `Cancelled`, `Refunded`, and `Payment error`, respectively. The result sets of the four queries should have no orders in common.

[1] http://demo.prestashop.com/en/?view=back

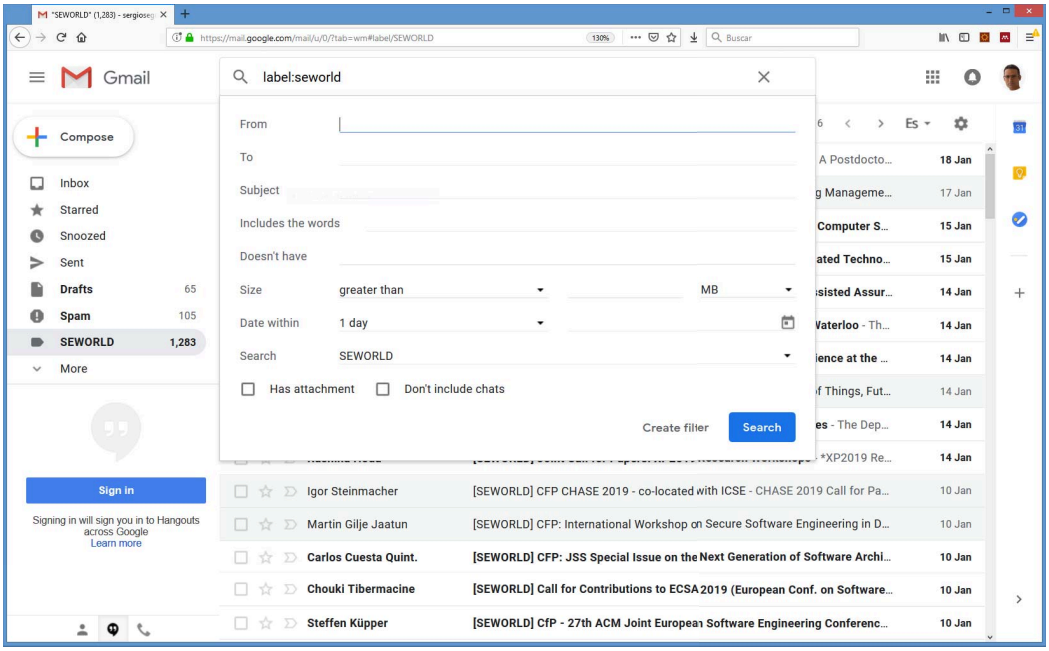

Figure 2. Gmail web client

- MR_5 (Complete partitions). List all the orders delivered to a given country (source test case). Next, repeat the query restricting the search to orders placed by an existing client, i.e., filter `New client = No` (follow-up test case 1). Then, repeat the query searching for orders placed by a new client, i.e., `New client = Yes` (follow-up test case 2). The result set of the source test case should include exactly the same orders as the union of the results returned in the follow-up test cases 1 (existing client) and 2 (new client).
- MR_6 (Partition difference). List all the orders. Next, repeat the query two times restricting the search to orders placed by an existing client (follow-up test case 1) and orders placed by a new client (follow-up test case 2), respectively. The difference between the result set of the source test case (all orders) and the result set of the follow-up test case 1 (orders placed by an existing client) should include exactly the same orders as the results returned in the follow-up test cases 2 (orders placed by a new client).

These metamorphic relations, and similar ones, could be easily identified in other e-commerce platforms (e.g., Magento [30]) and enterprise resource planning applications (e.g., OpenBravo [31]), where similar forms are used to query common domain objects such as customers, addresses, payments, invoices or refunds.

B. Gmail

Gmail [20] is a free web email service developed by Google with more than one billion active users worldwide. E-mails are classified using labels. By default, messages are classified using the labels *Inbox*, *Starred*, *Snoozed*, *Important*, *Chats*, *Sent*, *Drafts*, *All mail*, *Spam*, and *Bin*. In turn, the received messages (inbox) can be optionally arranged in five different tabs: *Primary*, *Social*, *Promotions*, *Updates*, and *Forums*. Gmail also supports e-mail search and a simple message classification system based on the use of custom labels and filters. Figure 2 shows the web user interface of Gmail showing the search form on top of the messages with the tag *SEWORLD*. We next show some of the metamorphic relations that can be easily identified as instances of the proposed MRPs.

- MR_1 (Input equivalence). Search for messages in the inbox greater than certain memory size in megabytes (e.g., `greater than 1MB`). Then, repeat the search expressing the memory amount in kilobytes (e.g., `greater than 1024KB`). Both queries should return exactly the same result set.
- MR_2 (Shuffling). Get all the messages in the inbox using the `Default` inbox type. Then, list all messages changing the inbox type to `Important first`, `Unread first`, and `Starred first`, respectively. The four queries should return the same messages, regardless of how they are arranged.
- MR_3 (Conjunctive conditions). Get all the messages in the inbox. Then, perform a search for those messages that `Has attachment`. The messages found in the second query should be a subset of the result set of the first query, where no attachment filter was specified.
- MR_4 (Disjunctive conditions). Get all the messages in the inbox from an email address a_1 (e.g., `from:jane@company.com`). Then, perform a search for those messages from a_1 or a_2, being a_2 a different

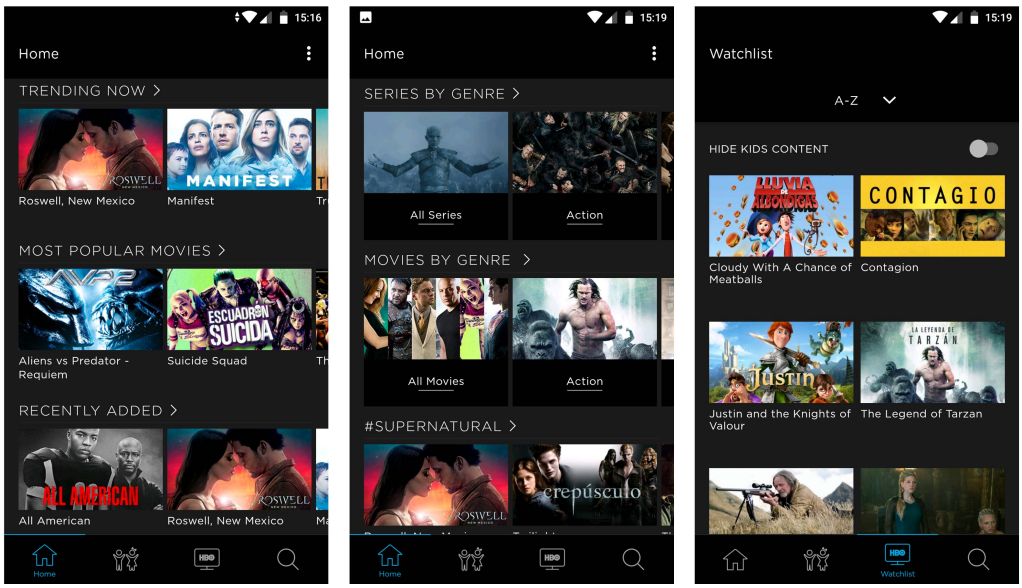

Figure 3. Some screenshots of the HBO mobile app (Spanish content)

email address (e.g., `from:jane@company.com OR from:peter@company.org`). The messages found in the first query should be a subset of the messages returned in the second query, where the search was broadened with a disjunctive condition.

- MR_5 (Disjoint partitions). List the messages `greater than` *x* MB. Next, search for those messages with `less than` *x* MB. The result sets of both queries should have no messages in common.
- MR_6 (Complete partitions). Search all the messages including a certain keyword. Next, repeat the search seven times restricting the search to `Inbox`, `Starred`, `Sent Mail`, `Drafts`, `Chats`, `Spam`, and `Bin`. The result set of the first search (source test case) should include exactly the same messages as the union of the results returned in the subsequent searches (follow-up test cases).
- MR_7 (Partition difference). Get all the messages in the inbox. Next, run five different queries to get all the messages in the inbox tabs `Primary`, `Social`, `Promotions`, `Updates`, and `Forums`. The difference between all the messages returned in the first query and those classified as social, promotions, updates and forums should be the messages included in the primary tab.

C. Home Box Office

Home Box Office (HBO) [21] is a popular premium cable and satellite television network, also operating as a subscription-based video streaming service, with more than 130 million subscribers worldwide. The platform content can be displayed in different devices including smart TVs, smartphones, tablets, and video game consoles. Figure 3 shows some screenshots of the HBO mobile app. As illustrated, it has a minimalistic design, with just a few input options for browsing the catalogue. The application mostly relies on a recommendation system for showing users new content based on their previous behaviour. Interestingly, it is also possible to identify metamorphic relations based on the proposed patterns in this type of application. These are some examples:

- MR_1 (Shuffling). Get content in the user watchlist ordered alphabetically (`A-Z`). Next, get the content in the user watchlist ordered by `Release date`. Finally, get the content ordered by the date in which it was added to the watchlist (`Recently added`). The three queries should return the same content, regardless of their ordering.
- MR_2 (Conjunctive conditions). Get all the content in the user watchlist. Then, get the content enabling the filter *hide kids content*. The result set of the second query (follow-up test case) should be a subset of the result set of the first query (source test case), where no filter was applied.
- MR_3 (Disjoint partitions). List the episodes of season 1 of a TV series. Next, list the episodes of seasons 2, 3, ..., n in subsequent queries. The result sets of all the queries should have no episodes in common.
- MR_4 (Complete partitions). List all movies (source test case). Then, perform different queries to get the movies of each available genre, including `Action`, `Sci-fi and fantasy`, `Comedy`, `Drama`, `Thriller`, `Family`, `Horror`, `Romance`, `Awarded`, `Documentary` and `Specials` (follow-up test cases). Intuitively, the union of the movies found in the follow-up test cases (removing possible duplicates) should be the same as those returned in the source test case.

These and many other metamorphic relations, instances of the proposed MRPs, could be easily identified in related video and music streaming platforms such as Amazon Prime

Video [32], Netflix [33] and Spotify [34].

V. Conclusions and future work

In this paper, we presented a catalogue of MRPs to assist on the identification of metamorphic relations in QBS. For a better understanding, patterns were formalized using relational algebra, a well-known formal approach for describing the semantics of query languages. To show the generalizability of the proposed patterns, we list some of the many metamorphic relations that can be derived from them in three different domains: an e-commerce platform, a web email service, and a video streaming app. The experience in this and previous papers reveals that patterns are extremely helpful to assist in the identification of metamorphic relations, guiding the testers in the search for relations with a certain shape. We trust that this paper serves as a helpful reference for the application of metamorphic testing in other QBS. Also, we hope that this paper encourages other researchers and practitioners to further investigate and exploit the benefits of using patterns in the context of metamorphic testing.

Future work may include the identification of new patterns as well as their use to test specific QBS. More importantly, we foresee a fruitful research line in exploiting the proposed patterns to automatically infer likely metamorphic relations in QBS using techniques like machine learning.

Acknowledgment

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project BELI (TIN2015-70560-R). We thank Zhi Quan Zhou for his discussions and comments on this paper.

References

[1] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[2] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *Software Engineering, IEEE Transactions on*, vol. 41, no. 5, pp. 507–525, May 2015.

[3] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology, Tech. Rep., 1998.

[4] S. Segura, G. Fraser, A. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, Sept 2016.

[5] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Software*, 2018. [Online]. Available: https://doi.org/10.1109/MS.2018.2875968

[6] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, pp. 4:1–4:27, Jan. 2018.

[7] Z. Q. Zhou, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of web search engines in the absence of an oracle," Department of Computer Science, The University of Hong Kong, Tech. Rep. TR-2007-06, 2007.

[8] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 221–243, Jun. 2012. [Online]. Available: http://dx.doi.org/10.1002/stvr.437

[9] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, March 2016.

[10] S. Segura, J. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of RESTful Web APIs," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1083–1099, Nov 2018.

[11] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic relations for enhancing system understanding and use," *IEEE Transactions on Software Engineering*, 2018. [Online]. Available: https://doi.org/10.1109/TSE.2018.2876433

[12] M. Lindvall, D. Ganesan, R. Ardal, and R. Wiegand, "Metamorphic model-based testing applied on nasa dat – an experience report," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2, May 2015, pp. 129–138.

[13] S. Segura, "Metamorphic testing: Challenges ahead (keynote speech)," in *Proceedings of the 3rd International Workshop on Metamorphic Testing (ICSE MET'18)*. New York, NY, USA: ACM, 2018, slides available at http://personal.us.es/sergiosegura/files/presentations/segura18-MET.pdf.

[14] J. Troya, S. Segura, and A. Ruiz-Cortés, "Automated inference of likely metamorphic relations for model transformations," *Journal of Systems and Software*, vol. 136, pp. 188 – 208, 2018.

[15] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2013*, Nov 2013, pp. 1–10.

[16] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels," *Software Testing, Verification and Reliability*, 2015. [Online]. Available: http://dx.doi.org/10.1002/stvr.1594

[17] C. Murphy, G. Kaiser, and L. Hu, "Properties of machine learning applications for use in metamorphic testing," Department of Computer Science, Columbia University, New York NY, Tech. Rep., 2008.

[18] H. García-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, 2nd ed. Pearson, 2009.

[19] "PrestaShop," accessed January 2019. [Online]. Available: https://www.prestashop.com/

[20] "Gmail," accessed January 2019. [Online]. Available: https://mail.google.com/mail

[21] "Home Box Office (HBO)," accessed January 2019. [Online]. Available: https://www.hbo.com/

[22] J. Ding, T. Wu, J. Q. Lu, and X. Hu, "Self-checked metamorphic testing of an image processing program," in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, June 2010, pp. 190–197.

[23] F. Su, J. Bell, C. Murphy, and G. Kaiser, "Dynamic inference of likely metamorphic properties to support differential testing," in *Proceedings of the 10th International Workshop on Automation of Software Test*, ser. AST '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 55–59.

[24] S. Nakajima and H. N. Bui, "Dataset coverage for testing machine learning computer programs," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, Dec 2016, pp. 297–304.

[25] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[26] J. M. Spivey, *The Z Notation: A Reference Manual*, 2nd ed. Prentice–Hall, 1992.

[27] ISO/IEC, "Information technology — Z formal specification notation — Syntax, type system and semantics," International Standard ISO/IEC 13568:2002, 2002.

[28] "Booking.com," accessed January 2019. [Online]. Available: https://www.booking.com

[29] "Paypal," accessed January 2019. [Online]. Available: https://www.paypal.com

[30] "Magento," accessed January 2019. [Online]. Available: https://magento.com/

[31] "Openbravo," accessed January 2019. [Online]. Available: http://www.openbravo.com/

[32] "Amazon Prime Video," accessed January 2019. [Online]. Available: https://www.primevideo.com

[33] "Netflix," accessed January 2019. [Online]. Available: https://www.netflix.com

[34] "Spotify," accessed January 2019. [Online]. Available: https://www.spotify.com