Testing Chatbots Using Metamorphic Relations

Josip Bozic(✉) and Franz Wotawa

Institute of Software Technology, Graz University of Technology, 8010 Graz, Austria
{jbozic,wotawa}@ist.tugraz.at

Abstract. Modern-day demands for services often require an availability on a 24/7 basis as well as online accessibility around the globe. For this sake, personalized software systems, called chatbots, are applied. Chatbots offer services, goods or information in natural language. These programs respond to the user in real-time and offer an intuitive and simple interface to interact with. Advantages like these makes them increasingly popular. Chatbots can even act as substitutes for humans for specific purposes. Since the chatbot market is growing, chatbots might outperform and replace classical web applications in the future. For this reason, ensuring correct functionality of chatbots is of high and increasing importance. However, since different implementations and user behavior result in unpredictable results, the chatbot's output is difficult to predict and classify as well. In fact, testing of chatbots represents a challenge because of the unavailability of a test oracle. In this paper, we introduce a metamorphic testing approach for chatbots. In general, metamorphic testing can be applied to situations where no expected values are available. In addition, we discuss how to obtain test cases for chatbots, i.e. sequences of interactions with a chatbot, in an according manner. We demonstrate our approach using a hotel booking system and discuss first experimental results.

Keywords: Metamorphic testing · Functional testing · Chatbots

1 Introduction

Virtual assistants are programs that realize the communication with a user in natural language. These programs, called chatbots [13], offer the advantage to resemble a natural and intuitive way of interaction. Usually, these programs comprehend information from a certain domain. In such way, the chatbot provides specific information in an often entertaining and anonymous manner [5]. Natural language interfaces (NLI) handle the communication between chatbot and user. On the side of the chatbots, this is usually implemented either by a set of rules or by means of natural networks. Here pre-specified patterns define boundaries of possible user interaction [20,24]. Since several studies predict the rise of the chatbot market in the future, addressing the functionality of these

C. Gaston et al. (Eds.): ICTSS 2019, LNCS 11812, pp. 41–55, 2019.
https://doi.org/10.1007/978-3-030-31280-0_3

systems becomes essential [2]. Until now, only a few testing approaches exist that check the correctness of chatbots (e.g. [4,22,23]).

Since the behavior of a chatbot might be difficult to predict, traditional testing approaches might not always provide an optimal solution. In testing, it's common practice to rely on test oracles in order to obtain test verdicts. However, problems may arise due to non-testable programs or because of the high amount of manual effort needed to draw a test conclusion. Difficulties with test oracles have been initially pointed to by Weyuker [25]. The author addressed the issue when test oracles are not available during testing. This can be due to multiple reasons, like high costs or complexity of implementations or the domain.

Driven by the same motivation, the concept of *metamorphic testing* (MT) has been introduced by Chen et al. [7]. In MT, the output of a system is not checked against an expected value but a *metamorphic relation* (MR). These describe a relation between the input data and the corresponding output. New test cases are inferred from previous ones according to the specified MRs. These inferred test cases are meant to detect errors that the former ones failed to do. MT is not meant only to detect errors in cases where no expected values are defined, but helps to reveal errors in the production phase as well. In addition to that, MT does not depend on a specific testing strategy; thus, it can be combined with other test selection approaches (see Sect. 4).

In this paper, we introduce a metamorphic testing approach for chatbots. The approach is defined by terms of MT and checks the functionality of a chatbot in an automated manner. Basically, the motivation of our approach is to guarantee functional correctness of a system under the absence of expected data. We provide the theoretical background and elaborate the implementation on a chatbot from the tourism industry. To our knowledge, this is the first adaptation of MT for testing of chatbots.

The remainder of the paper is structured as follows: Sect. 2 introduces metamorphic testing and its adaptation to testing of chatbots. Then, Sect. 2.1 describes the underlying test generation. Section 2.2 discusses the test execution as well as the underlying algorithm of the approach. The implementation is evaluated in Sect. 3, whereas related work is enumerated in Sect. 4. Finally, Sect. 5 concludes the work.

2 Metamorphic Testing for Chatbots

The bulk of this paper addresses two major fields: Metamorphic testing and one concrete manifestation of AI systems, namely chatbots. In our previous work in [4] we introduced a functional testing approach for chatbots, where the expected values were known in advance. The chatbot, a hotel booking system, is checked whether valid reservations are processed correctly. However, due to unpredictive behavior of such systems, MT addresses situations where only sparse or no knowledge exist about a system.

First, let's denote the basic principles of MT, as initially described in [7] and [17]. In contrast to other testing techniques, the basic principle for test generation differs in MT.

Definition 1. *Let f be an implemented function of a program P. R_f defines a property that must be valid for all inputs $x_1, x_2, ..., x_n$ over f, and the corresponding outputs $f(x_1), f(x_2), ..., f(x_n)$. The relation R_f represents a metamorphic relation (MR) over the input-output pairs, thus $R_f((x_1, f(x_1)), (x_2, f(x_2)), ..., (x_n, f(x_n)))$, where $n > 1$.*

Since no expected test value is available, properties have to be defined that must hold for all test inputs against a program, regardless of their value. However, conditions can be specified as part of a MR that put restrictions on test cases [17]. A metamorphic relation can represent any suitable type of relation, such as equalities, properties, constraints etc. The function of MR is twofold: (1) it serves as a guideline for generation of test inputs from existing data, and (2) acts as an output relation, which is checked after the test execution. In such way, MR replaces the test oracle by comparing the input-output pairs from test cases. We call this comparison a *metamorphic check* (MC). A metamorphic check fails in case that the input-output pair satisfies a metamorphic relation.

Definition 2. *An initial source test case x_1 that does not trigger an error or vulnerability in P, is a successful test case. A follow-up test case x_2 is inferred from x_1 according to a selection criteria, as specified by R_f. The pair of the source and follow-up test case can be defined as a metamorphic test case.*

Usually, the source test case represents a random value or is derived from partial knowledge about the domain of P. All new test cases are derived from the successful test cases. For the source test case x_1 and its output $f(x_1)$, new test cases will be generated based on the input-output pair $(x_1, f(x_1))$. During test execution, the applied MRs must hold for input-output pairs of both, source and the follow-up, test cases. If the MR does not hold for a certain metamorphic test case [17], then we conclude that a fault or inconsistency is detected in P.

In general, three tasks constitute a MT-based approach, regardless of the domain:

1. Definition of metamorphic relations
2. Test generation
3. Test execution

In the following sections, we will address each of the challenges individually. For all three cases, the adaptation of MT for chatbots has to be addressed in detail.

2.1 Metamorphic Test Generation

We describe our MT approach for testing of chatbots and evaluate it on a real example. The tested chatbot is a hotel booking system that was introduced in [4]. In this paper, we will omit technical details about the implementation but provide a few details about its functionality. The agent is implemented in Dialogflow [1] and communicates with the user by sending and receiving messages in natural language. Since the goal of the chatbot is to make a hotel booking, the chatbot demands some information in order to finalize a reservation. On the side of the chatbot, this information is hard coded in form of eight mandatory parameters. During communication, the chatbot will ask the user to provide this information. Also, we assume that the chatbot comprises the natural processing and necessary data. A typical communication flow, initiated by the user, does look as follows.

```
I want to book a room in Vienna.
- Do you want a hotel, hostel or apartment?
book a hotel
- Which one do you prefer?
Fairmont
- When do you want to start your stay?
I want to check-in today.
- How many adults?
1 adult
- How many children?
2 children
- How many nights?
1 night
- How many stars do you prefer?
i want 5 stars
- Let me sum up! Location: Vienna, Name: Fairmont, Check-In:
  2019-05-20, Accommodation: hotel, People: 1 adults, 2 children,
  Nights: 1, Stars: 5 stars
```

In case that the user does not provide a specific information, the chatbot will continue to ask for clarification. Thus, no reservation will be made. On the other hand, the behavior of the user can be unpredictable to a certain degree as well. For example, the user can make an inquiry by rephrasing sentences or providing multiple information in one message. This fact will play a role in the test generation process. The chatbot is expected to recognize different types of input utterances. In fact, the chatbot understands additional messages (like `exit` or `abort`), but in our example, we ignore such situations. However, regardless of the user input, the chatbot demands information about the following parameters.

```
$location, $venue-type, $venue-title, $checkin-date, $adult,
$child, $night, $star
```

In our previous work, we tested whether the chatbot makes a valid reservation. However, in this paper, the functionality is tested from another perspective. From the starting point of view, the chatbot is unpredictable for the user. However, the user is unpredictable in front of the chatbot as well. In fact, we assume that the user initially knows the information that she or he intends to submit. Also, the intention of the user is to make a valid reservation with this information. That is, the information in a reservation must not deviate from the provided values. Additionally, according to practice in MT, we assume that some information about the domain is known in advance. One sequence of messages from the point of the user is considered to be one test case. We suppose that the input values for the source test case are denoted as I_s, whereas the follow-up values are given as I_f. Also, their outputs are defined as O_s and O_f, respectively. The obtained output serves as an indicator for the test verdict. The output differs for individual inquiries and available information. For example, different numerical inputs will result in different registration information. The type of responses, however, will remain the same.

As can be seen Table 1, I_s consists of eight actions that will be submitted as individual requests. The individual actions themselves comprise several terms and numerical values that are considered keywords. As will be described below, these play an important role for further test generation. After every request, the chatbot is programmed to give a corresponding response. During testing, we record all responses for every submitted action. On the right side of the table the chatbot's mandatory parameters are enumerated. The chatbot processes the individual actions and, if possible, assigns provided values to their parameter.

Table 1. Source test case

#	Action	Parameter
0:	I want to book a room in Vienna	`$location`
1:	book a hotel	`$venue-type`
2:	Fairmont	`$venue-title`
3:	I want to check-in today	`$checkin-date`
4:	1 adults	`$adult`
5:	2 children	`$child`
6:	1 night	`$night`
7:	i want 5 stars	`$star`

Now, we define the following properties of the chatbot as MRs. As already mentioned, these relations must be valid for test inputs and outputs. The source test case from Table 1 encompasses sentences in natural language. However, several keywords are emphasized that will be addressed by the MRs. The MRs focus on input generation for chatbots, as follows:

MR1: Replace keywords with synonyms: In order to infer I_f from I_s, they keywords are replaced by synonyms. Although, for every follow-up test case, keywords are changed in only one action. The obtained O_s serves as a reference that is compared to the obtained O_f. When applying synonyms, we expect the same behavior from the chatbot, thus $O_s = O_f$ must be valid.

MR2: Change numerical values: This relation instructs to change numerical values for individual actions. The outputs of O_s and O_f should be equal, with the exception of the numerals from the final reservation.

MR3: Omit words: This relation demands that keywords are omitted, thus providing insufficient information. The chatbot should recognize the difference, thus resulting in different responses. MR3 is valid in case that $O_s \neq O_f$ for every I_f.

MR4: Replace keywords with unrelated terms: This relation is similar to MR1 but instead of synonyms, keywords are exchanged with unrelated terms. Therefore, the corresponding outputs must never resemble each other. Therefore, the expected outputs equal $O_s \neq O_f$.

MR5: Change order of actions: The order of actions in I_s is changed in the follow-up test case I_f. Since the tester does not know about the SUT's behavior, the obtained outputs are expected to differ, thus $O_s \neq O_f$. (Although, a valid reservation might still be accomplished accidentally by a valid random combination.)

For every MR, a grammar is constructed that provides the building blocks for test generation. For example, the grammar for MR1 is defined in the standard Backus-Naur form (BNF).

```
<sentence>::= I want to <book> a <room> in <place>
<book>::= book | order | take | prefer
<room>::= room | place | hotel | hostel | apartment
<place>::= Vienna | Prague | Madrid
<accomodation>::= hotel | hostel | apartment
<venue>::= Sacher | Fairmont | Elmhurst
<days>::= in two days | yesterday | tomorrow | today
<adults>::= <number> <guests>
<guests>::= guests | people | humans | adults
<youngling>::= <number> <children>
<children>::= children | kids
<nights>::= <number> <sleep>
<sleep>::= nights | days
<rating>::= <number> <points>
<points>::= stars | ratings
<number>::= 1 | 2 | 3 | 4 | 5
```

Formal grammars are defined by finite sets of terminal and nonterminal symbols. In the above example, nonterminal symbols resemble keywords in our approach, whereas terminals represent concrete values. The expression, i.e. a sequence of symbols, guides the generation of an action by assigning concrete values according to MR. Some terminals in the grammar are set manually by checking the supported library from Dialogflow. In order to generate meaningful inputs, we use a modified version of Grammar Solver [3]. The implementation applies MRs and traverses through the BNF structure, thereby constructing new user inputs. In fact, this process can be subtracted as a mutation process. MRs proscribe unique mutations to individual actions, thus generating modified, i.e. follow-up, test cases. So, the final shape of a follow-up test cases is determined by the mutation operation and the source test case.

Table 2. Follow-up test case for MR4

#	Action
0:	I want to book a room in nowhere
1:	book a hotel
2:	Fairmont
3:	I want to check-in today.
4:	1 adults
5:	2 children
6:	1 night
7:	i want 5 stars

For example, the follow-up test case I_f in Table 2 depicts the case where MR4 is applied to I_s from Table 1. In fact, every inferred test case will differ from its original by adding one change. In such way, we want to check how minimal changes affect the overall testing result. We obtain a diverse test case by applying mutations to just one initial test. By doing so, our intention is to obtain tests that are able to lead to states during execution that are not covered by the implementation or the specification. After the mutations from all MRs have been applied to the source test case, we initiate the execution process.

2.2 Metamorphic Test Execution

After the individual MRs have been specified, the test generation process can begin. The algorithm, **BotMorph**, as shown in Algorithm 1, implements our entire test generation and execution approach. The process starts with the initial source test case, I_s, from Table 1. Its individual actions are manually defined and represent a sequence that ultimately leads to a specific reservation. I_s is executed and the chatbot's source output, O_s, is recorded. This represents an important step that substitutes the test oracle: The comparison of O_s to the result from the follow-up test case, O_f, will determine the test verdict.

Then, the MRs are applied in optional order to derive follow-up test cases. The new test case is executed against the SUT and the output is recorded. From the technical point of view, obtained values from Grammar Solver are assigned to HTTP requests and submitted to the chatbot system on the fly. In return, its HTTP responses are processed and handled accordingly. The MR checks are done after the final O_f of a test case is received. If no vulnerability is triggered, i.e. if the MC fails, then the test case is added to the list of successful tests V. Otherwise, in case that an issue has been detected, the test will be disregarded. The process continues until all MRs have been applied and checked. Afterwards, the same process is restarted but with new source tests from the successful ones.

In theory, the process can continue until every symbol has occurred for every combination of values for all MRs. It remains the task of the tester to define the termination condition.

Algorithm 1. BotMorph – Metamorphic test generation and execution algorithm for chatbots

Input: Program P, set of metamorphic relations $MR = \{\mathtt{mr}_1, \ldots, \mathtt{mr}_n\}$, source test case $I_s = \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$, grammar G and a function $\Phi = (MR, I_s, G) \mapsto I_f$ that infers follow-up test cases.
Output: Metamorphic test set $TS = \{\mathtt{I}_{f1}, \ldots, \mathtt{I}_{fi}\}$ where each $I_{fi} = \{\mathtt{a}_0, \ldots, \mathtt{a}_n\}$ and a list with source test cases $V = \{\mathtt{v}_1, \ldots, \mathtt{v}_n\}$.

```
 1: $TS = \emptyset$
 2: $O_s = exec(I_s, P)$
 3: $V = V \cup \{(I_s, O_s)\}$
 4: for $v \in V$ do
 5:     while $MR.hasNext()$ do
 6:         $I_f = generate(\Phi, mr_n, G, I_s)$                ▷ Generate test case
 7:         $TS = TS \cup \{I_f\}$
 8:         for $I_f \in TS$ do
 9:             $res(I_f) = FAIL$
10:             $O_f = exec(I_f, P)$                             ▷ Execute test case
11:             if check$((I_s, O_s), (I_f, O_f), mr_n)$ fails then      ▷ Execute MC
12:                 $res(I_f) = PASS$
13:                 $V = V \cup \{(I_f, O_f)\}$                  ▷ Save successful test case
14:             else
15:                 $res(I_f) = FAIL$
16:             end if
17:         end for
18:     end while
19: end for
```

3 Evaluation

For the evaluation we generated several test sets from the initial test case. The source test case I_s serves as the starting point for all inferred I_f's. The defined MRs serve as test guidelines as well as the basis for conducted metamorphic checks. We execute the procedure from M1 in a descending order. Since the individual actions from I_s differ from each other, the MRs will cause different results with regard to number and shape of test cases. For example, a_0 contains three keywords, whereas a_1 comprehends just one. When applying MR1, synonyms will be created for every keyword separately (with the exception of the check-in date). For example, in case of a_0, the mutation is applied only for one keyword at the time. Thus, the number of resulting follow-up's is multiplied by each keyword and the grammar values. On the other hand, a_1 will provide only one follow-up test case per mutation. Also, each MR is applied individually on just one action, so the new test case differs only in one change from its original (the exception being MR5). In such way we want to observe the testing behaviour by adding just small changes.

In total, we generated 104 test cases that were submitted against the booking chatbot. Table 3 depicts all results that have been achieved.

Table 3. Results for metamorphic test cases

MR	#total	#pass	#fail
MR1	60	56	4
MR2	16	16	0
MR3	10	5	5
MR4	10	5	5
MR5	8	8	0

Since the SUT behaves different for each input, we will analyse the results with regard to every MR.

MR1: We added several synonyms for keywords and encountered a few discrepancies with Dialogflow. The initial booking request action was successful in all cases. Thus, the chatbot was able to recognize the intent and finalize the reservation even when the client reformulates the request. A failing test case was encountered with a_{4-6} and a_8, where synonyms like "guests" instead of "adults", "kids" instead of "children" and "ratings" could not be recognized by the chatbot.

MR2: This MR is unique among the relations because it affects only actions with numerical values. That is, only four out of eight inputs are addressed by the mutation. It can be concluded that changing the number in an action does not cause any failures. The chatbot succeeds with generating a reservation with different values. Therefore, all tests have been successful.

MR3: Omitting words causes a lot of confusion in the chatbot. Initiating a reservation with a_0 is possible even without the explicit mentioning of "book" or "room". However, if no city name is provided, the client cannot proceed further. We conclude that the explicit statement of a city name at the beginning of the communication is mandatory. Also, if no accommodation type, check-in dates and stars are provided, then the reservation fails. The chatbot keeps insisting for the information and ignores further user requests. For actions with numerals, we kept the numbers but omitted the words to their right. Interestingly, a valid information is made when providing just numerals without "adults", "children" or "nights". It seems that the chatbot assumes that a client always provides information in a strict order. Thus, it assigns the information to parameters without further clarification. However, this seems not to be the case when "stars" is omitted.

MR4: Replacing keywords with nonrelated terms triggered strange behaviour on side of the chatbot. For example, formulating a_0 into "I want to exchange a pod in nowhere" (three follow-up test cases, one for every keyword) resulted in a valid registration! If we compare the behaviour to MR3, the chatbot seems to accept every value for city name as a valid one. This is an important implementation flaw in the chatbot. Actions a_{2-4} are processed as expected by the SUT. On the other hand, "alien" and "kobolds" seem to be a valid substitute for "adults" and "children" in a_5 and a_6, for unknown reasons. However, a_7 and a_8 have been rejected, as demanded by the relation.

MR5: This MR is unique since it requests a reordering of actions in a test case. This means that concrete values will not differ between I_s and I_f. We adapted a randomized approach that resulted in eight test cases with different sequences of actions. As expected, every O_f from SUT differs from O_s. The chatbot's response depends on the first user inquiry. Usually the SUT keeps asking for a certain information and remains in a specific queue. If the requested information is received in the meanwhile, then it switches to another queue. Also, if the chatbot encountered unexpected input (from its point of view), then unexpected responses were sent. This included an empty text or unrelated sentences (e.g. "Talk to you soon!").

Since we applied MT for functional testing purposes, our intention was not to exploit a SUT. Since a tester does not know about the inner workings of a SUT and its available information, some guessing is needed. The tester can assume that some keywords will be understood by the chatbot, others can be completely avoided. In our approach, unintended behaviour indicates that (1) an implementation flaw or oversight or (2) insufficient information on side of the Dialogflow implementation. Actually, here both observations affirm known advantages of MT, namely the detection of verification and validation defects, respectively [8]. In the first case, we can assume that the chatbot did behave to its implementation or incomplete specification. However, it failed to cover cases, which it should be able to understand. Also, the assignment of information to a specific parameter without clear indication can be considered a drawback as well. Finalizing a reservation with unsuitable data (e.g. fictional city names or substitutes) is even worse.

On the other hand, the chatbot's permanent insistence on a specific information at a certain point during the conversation indicates that a strict order must be followed. As already proven in [4], the hotel booking chatbot insists that the first and last inquiries must be of a specific type. The sequence of other information is optional.

```
$location
$venue-type
$venue-title
$checkin-date
$adult
$child
$night
$star
```

The approach in that work demonstrated that the chatbot concluded reservations even with missing information in a nonintuitive manner. This means that the chatbot does something it should not. The same observation was proven when testing against MR3. On the other hand, MT detected that the SUT does not what it should do (e.g. with MR1). Also, meaningless input is wrongly associated with an intent with MR4. We assume that the reason why unrelated terms are still "understood" is due to the fact that the chatbot follows its strict order and ignores other information ("stars" being an exception). The recognition of terms depends upon the pre-specified entries in the chatbot's entity database. A different set of values and intents would likely result in different behaviour of the system. Interestingly, some natural language input does disturb the chatbot as well. Adding a "I want to" to mandatory information does not match the chatbot's intent. It seems that the SUT follows a minimalistic approach.

In general, we conclude that the use of metamorphic test cases succeeds to trigger defects in natural language systems. In addition to that, it provides some clarification about possible reasons with regard to diverse inputs. Also, it uncovers clues about inner workings of the chatbot by relying on a small set of metamorphic relations. For all these reasons, we consider the presented approach as a good starting concept for further actions.

4 Related Work

Research that correlates with our work includes metamorphic testing and chatbot testing. Works that focus on the first topic are often conducted on industrial and online applications. Also, these approaches often interact with external implementations and resources. On the other hand, literature that focuses on testing of chatbots, is sparse. To our knowledge, no work deals with the adaptation of MT to chatbots.

4.1 Metamorphic Testing

MT was introduced in [7] and the idea was elaborated on several examples of numerical problems. Comprehensive surveys about papers that deal with MT can be found in [17] and [8].

The preliminary work that addressed oracle-free programs is discussed in [25]. Here the notion of *pseudo-oracles* is used, i.e. external programs that substitute a real test oracle. Additional programs are implemented and tests are run against both the original and the substitute program. If the results match, then the results of the original program are considered to be valid.

In [18] and [16] the authors elaborate a MT-based approach for testing of *feature models* (FM) and *variability models* (VM). Such models encompass the definition of specific products by means of features or configurations, respectively. MRs are defined between models and their inferred products, and a test generator. New models are generated from previous ones by adding mandatory features. In this way, many models and valid configurations can be generated automatically.

[26] addresses testing of implementations that use ML classification algorithms. The authors derive MRs for the individual algorithms and test a real-world ML application framework. Individual MRs define properties that are

necessary to be valid for a classifier algorithm. Finally, they claim that their MT-based approach can be used against any SUT that uses ML. This work is evaluated further in [15].

In [28] a MT-based framework is proposed for testing of autonomous driving systems that rely on Deep Neural Networks (DNN). Another work that elaborates MT for testing of autonomous systems is given in [29]. Here MT relies on deep learning models and comes in combination with fuzzing. The authors evaluate their approach on the mission-critical LiDAR Obstacle Perception system, with promising results.

Additional adaptations of MT to AI systems that include autonomous systems and DNN are elaborated in [12] and [21], respectively. Also, [10] discusses the use of MT for debugging of ML applications.

The authors of [9] address the applicability of MT to cybersecurity. In this approach, MRs are defined in order to test for web failures. Several real-world obfuscators have been tested by using a small test suite. The conducted evaluation indicates that several bugs were found.

Additionally, [19] applies MT to test Web Application Programming Interfaces (APIs).

[6] discusses an implementation of MR for testing of event sequences for business processes. The approach, MTES, defines metamorphic relations for fault-detection purposes. The authors conclude that more diverse inputs result in higher detection capabilities.

Also, works exist that address MT for machine translators (e.g. [27]). A MR applies translations to different sentences and checks for semantic (dis)similarity between languages. Inputs are generated that are meant to be homologous in different languages. The authors discuss the quality of translators and their translation errors. In contrast to our work, they validate the produced inputs from the translator and ignore the SUT that processes these inputs.

4.2 Testing of Chatbots

In contrast to the plentiful number of MT-related papers, only a few papers address testing of chatbots.

For example, [23] introduces PARADISE, a system that deals with the evaluation of conversational systems. Different dialogue strategies are compared and checked against a performance function. In this way, the correctness of an answer is determined. However, in our work, we concentrate on the user's side without having an insight in expected results.

[22] that deals with testing of conversational systems with regard to functionality. A testing system emulates a user that interacts with the chatbot. A large amount of pre-defined user inputs is submitted to the chatbot in an automated manner. The tested system itself, CognIA, represents a financial advisor. During exhaustive communication, specific metrics are used that validate the chatbot's responses with regard to submitted inputs.

In our previous work [4], we introduced an approach for functional testing of a hotel booking chatbot. We applied AI planning for generation of test cases and compared the chatbot's output to an expected result in an automated manner.

Techniques that resemble our test case generation technique are given in [14] and [11]. They discuss a technique for conversational input generation for chatbots. The idea behind the proposed technique is to change valid user inputs by paraphrasing them. Different techniques are used for this sake in order to retrieve a divergent input from the original one. The changes encompass lexical substitutions, as well as non-native preposition errors and native colloquial phrasing, respectively. An external source is used for retrieving original synonyms. Finally, the resulting framework tests the robustness of the system by checking on the input variations. The evaluations, based on a chatbot and a flight booking system, indicate that several weaknesses have been encountered. The obtained results indicate that the SUT does not always recognize an intent when words are replaced with synonyms. Consequently, it might be claimed that our results from MR1 complement the observations from their approach. However, the reasons for these observations should be investigated in more detail. On the other hand, the main difference represents the fact that their technique relies on known expected values. This stands in stark contrast to a MT-based approach. In our case, the use of multiple MRs yield more insight into the inner workings of the SUT.

5 Conclusion and Future Work

In this paper, we addressed an emerging issue in software testing, namely testing of AI systems. The nature of such systems often impedes a straight-forward approach. Especially systems that communicate in natural language might become more important in the future. Because of that, testing of these systems becomes an essential task to ensure correct functionality. In this paper we introduced a metamorphic testing approach for testing of chatbots. Instead of models of existing systems, it relies on metamorphic relations. Since the output of an AI system is difficult to predict, the approach introduces metamorphic checks instead of traditional test oracles. We presented a program that combines metamorphic test case generation and execution for functional testing of chatbots. In the aftermath, the approach is successfully evaluated on a chatbot system from the tourism industry.

The obtained results indicate that the metamorphic approach is able to detect unexpected behaviour in a system. Although the chatbot did succeed to ensure functionality in some cases, other test cases triggered situations, which the SUT failed to handle correctly. Consequently, the reasons for these issues must be addressed separately.

Finally, we claim that our approach can be used for functional testing of chatbots. In addition to that, metamorphic testing can be used for testing of non-functional properties as well. Precisely, MT-based approaches can be applied under circumstances where expected behaviour is difficult or impossible to determine. In the future, we plan to extend our approach by improving the test generation. MRs can be easily added to the existing ones. Thus, a more diverse test suite will be obtained. Also, using the MT approach against other chatbot systems remains an open challenge. Eventually, it would be interesting to compare the MT-based approach to other chatbot testing techniques and tools.

Acknowledgement. The research presented in the paper has been funded in part by the Cooperation Programme Interreg V-A Slovenia-Austria under the project AS-IT-IC (Austrian-Slovenian Intelligent Tourist Information Center). We thank our colleague Oliver A. Tazl, who helped us with the implementation of the SUT in [4], which was used in this work. In addition to that, we want to thank the anonymous reviewers for their constructive feedback, which we addressed in the paper.

References

1. Dialogflow. https://dialogflow.com/. Accessed 11 Dec 2018
2. Gartner Top Strategic Predictions for 2018 and Beyond. https://www.gartner.com/smarterwithgartner/gartner-top-strategic-predictions-for-2018-and-beyond/. Accessed 07 May 2018
3. Grammar-solver. https://github.com/bd21/Grammar-Solver. Accessed 13 July 2018
4. Bozic, J., Tazl, O.A., Wotawa, F.: Chatbot testing using AI planning. In: Proceedings of the International Conference on Artificial Intelligence Testing (AITest) (2019)
5. Brandtzæg, P.B., Følstad, A.: Why people use chatbots. In: Proceedings of the 4th International Conference on Internet Science (INSCI 2017) (2017)
6. Chen, J., Wang, Y., Guo, Y., Jiang, M.: A metamorphic testing approach for event sequences. PLoS ONE **14**(2), e0212476 (2019)
7. Chen, T.Y., Cheung, S.C., Yiu, S.M.: Metamorphic testing: a new approach for generating next test cases. Technical report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong (1998)
8. Chen, T.Y., et al.: Metamorphic testing: a review of challenges and opportunities. ACM Comput. Surv. (CSUR) **51**(1), 4 (2018)
9. Chen, T.Y., et al.: Metamorphic testing for cybersecurity. Computer **49**(6), 48–55 (2016)
10. Dwarakanath, A., et al.: Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018) (2018)
11. Guichard, J., Ruane, E., Smith, R., Bean, D., Ventresque, A.: Assessing the robustness of conversational agents using paraphrases. IEEE, University College Dublin (2019)
12. Lindvall, M., Porter, A., Magnusson, G., Schulze, C.: Metamorphic model-based testing of autonomous systems. In: Proceedings of the 2nd International Workshop on Metamorphic Testing (MET 2017) (2017)
13. Mauldin, M.L.: ChatterBots, TinyMuds and the turing test: entering the loebner prize competition. In: AAAI 1994 Proceedings of the Twelfth National Conference on Artificial Intelligence, vol. 1, pp. 16–21 (1994)

14. Ruane, E., Faure, T., Smith, R., Bean, D., Carson-Berndsen, J., Ventresque, A.: BoTest: a framework to test the quality of conversational agents using divergent input examples. In: Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion (IUI 2018 Companion) (2018)
15. Saha, P., Kanewala, U.: Fault detection effectiveness of metamorphic relations developed for testing supervised classifiers. In: Proceedings of the International Conference on Artificial Intelligence Testing (AITest) (2019)
16. Segura, S., Durán, A., Sánchez, A.B., Le Berre, D., Lonca, E., Ruiz-Cortés, A.: Automated metamorphic testing of variability analysis tools. Softw. Test. Verif. Reliab. **25**(2), 138–163 (2015)
17. Segura, S., Fraser, G., Sánchez, A.B., Ruiz-Cortés, A.: A survey on metamorphic testing. IEEE Trans. Softw. Eng. **42**(9), 805–824 (2016)
18. Segura, S., Hierons, R.M., Benavides, D., Ruiz-Cortés, A.: Automated test data generation on the analyses of feature models: a metamorphic testing approach. In: Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation (2010)
19. Segura, S., Parejo, J.A., Troya, J., Ruiz-Cortés, A.: Metamorphic testing of RESTful web APIs. IEEE Trans. Softw. Eng. **44**(11), 1083–1099 (2018)
20. Shawar, B.A., Atwell, E.: Using corpora in machine-learning chatbot systems. Int. J. Corpus Linguist. **10**, 489–516 (2005)
21. Tian, Y., Pei, K., Jana, S., Ray, B.: DeepTest: automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering (2018)
22. Vasconcelos, M., Candello, H., Pinhanez, C., dos Santos, T.: Bottester: testing conversational systems with simulated users. In: Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems (IHC 2017) (2017)
23. Walker, M.A., Litman, D.J., Kamm, C.A., Abella, A.: PARADISE: a framework for evaluating spoken dialogue agents. In: Proceedings of the 35th Annual General Meeting of the Association for Computational Linguistics, ACL/EACL 1997 (1997)
24. Wallace, R.S.: The elements of AIML style. In: ALICE A.I. Foundation (2003)
25. Weyuker, E.: On testing non-testable programs. Comput. J. **25**(4), 465–470 (1982)
26. Xie, X., Ho, J.W.K., Murphy, C., Kaiser, G., Xu, B., Chen, T.Y.: Testing and validating machine learning classifiers by metamorphic testing. J. Syst. Softw. **84**(4), 544–558 (2011)
27. Yan, B., Yecies, B., Zhou, Z.Q.: Metamorphic relations for data validation: a case study of translated text messages. In: Proceedings of the 4th International Workshop on Metamorphic Testing (MET 2019) (2019)
28. Zhang, M., Zhang, Y., Zhang, L., Liu, C., Khurshid, S.: DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018) (2018)
29. Zhou, Z.Q., Sun, L.: Metamorphic testing of driverless cars. Commun. ACM **62**(3), 61–67 (2019)