LICENTIATE THESIS

Test Automation for Grid-Based Multiagent Autonomous Systems

Sina Entekhabi

Halmstad University | School of Information Technology
LicentiateThesis | Halmstad University Dissertations no. 116

Test Automation for Grid-Based Multiagent Autonomous Systems

Sina Entekhabi

LICENTIATE THESIS | Halmstad University Dissertations no. 116

Test Automation for Grid-Based Multiagent Autonomous Systems

Halmstad University Dissertations no. 116
ISBN 978-91-89587-49-6 (printed)
ISBN 978-91-89587-48-9 (pdf)
Publisher: Halmstad University Press, 2024 | www.hh.se/hup
Printer: Media-Tryck, Lund

Abstract

Traditional software testing usually comes with manual definitions of test cases. This manual process can be time-consuming, tedious, and incomplete in covering important but elusive corner cases that are hardly identifiable. Automatic generation of random test cases emerges as a strategy to mitigate the challenges associated with the manual test case design. However, the effectiveness of random test cases in fault detection may be limited, leading to increased testing costs, particularly in systems where test execution demands substantial resources and time. Leveraging the domain knowledge of test experts can guide the automatic random generation of test cases to more effective zones. In this thesis, we target quality assurance of multiagent autonomous systems and aim to automate test generation for them by applying the domain knowledge of test experts.

To formalize the specification of the domain expert's knowledge, we introduce a small Domain Specific Language (DSL) for formal specification of particular locality-based constraints for grid-based multiagent systems. We initially employ this DSL for filtering randomly generated test inputs. Then, we evaluate the effectiveness of the generated test cases through an experiment on a case study of autonomous agents. Applying statistical analysis on the experiment results demonstrates that utilizing the domain knowledge to specify test selection criteria for filtering randomly generated test cases significantly reduces the number of potentially costly test executions to identify the persisting faults.

Domain knowledge of experts can also be utilized to directly generate test inputs with constraint solvers. We conduct a comprehensive study to compare the performance of filtering random cases and constraint-solving approaches in generating selective test cases across various test scenario parameters. The examination of these parameters provides criteria for determining the suitability of random data filtering versus constraint solving, considering the varying size and complexity of the test input generation constraint. To conduct our experiments, we use QuickCheck tool for random test data generation with filtering, and we employ Z3 for constraint solving. The findings, supported by observations and statistical analysis, reveal that test scenario parameters impact the performance of filtering and constraint-solving approaches differently. Specifically, the results indicate complementary strengths between the

two approaches: random generation and filtering approach excels for the systems with a large number of agents and long agent paths but shows degradation in larger grid sizes and stricter constraints. Conversely, constraint solving approach demonstrates robust performance for large grid sizes and strict constraints but experiences degradation with increased agent numbers and longer paths.

Our initially proposed DSL is limited in its features and is only capable of specifying particular locality-based constraints. To be able to specify more elaborate test scenarios, we extend that DSL based on a more intricate model of autonomous agents and their environment. Using the extended DSL, we can specify test oracles and test scenarios for a dynamic grid environment and agents having several attributes. To assess the extended DSL's utility, we design a questionnaire to gather opinions from several experts and also run an experiment to compare the efficiency of the extended DSL with the initially proposed one. The questionnaire results indicate that the extended DSL was successful in specifying several scenarios that the experts found more useful than the scenarios specified by the initial DSL. Moreover, the experimental results demonstrate that testing with the extended DSL can significantly reduce the number of test executions to detect system faults, leading to a more efficient testing process.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Associate Professor Wojciech Mostowski and Professor Mohammad Reza Mousavi, for their invaluable guidance, support, and encouragement throughout the duration of this thesis. Their expertise, dedication, and unwavering commitment have been instrumental in shaping this work and steering me in the right direction.

Furthermore, I would like to extend my appreciation to Professor Jan Tretmans for his willingness to invest his time and expertise in mentoring me. I should also thank Thomas Arts for all of his support and contributions from Quviq company side. Moreover, I am thankful for the continuous support and encouragement provided by the faculty and staff of Halmstad University, especially Veronica Gaspes, whose contributions have been invaluable in facilitating the completion of this thesis.

Lastly, I would like to express my deepest gratitude to my family and friends for their unwavering love, encouragement, and understanding throughout this journey.

List of Papers

The following papers are included in this thesis.

PAPER I: **Locality-based Test Selection for Autonomous Agents**
Sina Entekhabi, Wojciech Mostowski, Mohammad Reza Mousavi, Thomas Arts. **International Conference on Testing Software and Systems (ICTSS 2021)**, pp 73–89, DOI: 10.1007/978-3-031-04673-5_6.

PAPER II: **Automated and Efficient Test-Generation for Grid-Based Multiagent Systems**
Sina Entekhabi, Wojciech Mostowski, Mohammad Reza Mousavi. **ACM Transactions on Software Engineering and Methodologies Journal (TOSEM 2023)**, Volume 33, Issue 1, Article No.: 12, pp 1–32, DOI: 10.1145/3613536.

PAPER III: **Domain Specific Language for Testing Grid-based Multiagent Systems**
Sina Entekhabi, Wojciech Mostowski, Mohammad Reza Mousavi. **Manuscript** - *to be submitted to International Symposium of Software Reliability Engineering (ISSRE 2024).*

Contents

Abbreviations

ART	Adaptive Random Testing
AV	Automated Vehicles
CRV	Constraint Random Verification
DSL	Domain Specific Language
MBT	Model Based Testing
PBT	Property Based Testing
SMT	Satisfiability Modulo Theory
SUT	System Under Test

List of Figures

1. Introduction

1.1 Motivation

Software testing is a part of the software development life-cycle aiming to assure the software quality before making it available to the end users. Testing a system is usually carried out by executing and monitoring the system in particular scenarios and evaluating if the executed behavior in each scenario is acceptable. This process typically accounts for over half of the software development costs [1]. In traditional software testing, the test cases are designed manually, which is time-consuming and tedious. In the manual design, some elusive important corner cases to test can also be missed when they are hard to identify. In addition, maintenance of such test cases can be challenging since they should be manually re-designed when the system requirements change. Such problems can be mitigated by the automated generation of test cases at a lower cost, e.g., by utilizing **Model-based Testing (MBT)** [2] or **Property-based Testing (PBT)** [3] methods once the model is in place. In this thesis, we aim to automate test case generation for the systems of our domain, hoping to assure their quality with less cost and more precision.

1.2 Application Domain

The work presented in this thesis was developed in the context of *SafeSmart* project [4] and followed its goals in investigating *Safety of Connected Intelligent Vehicles in Smart Cities*. The project includes investigation on vehicle control, Vehicle_to_X (V2X) communication, sensing and localization of the objects on the road, and testing and integration of the communicating vehicles. The focus of this thesis is to contribute to the testing and integration part of the project. Dense urban traffic is the context of the project, and simulation is the main method of development and validation. The application of model-based techniques for simulation-based testing in this domain is our main goal. To tackle safety assurance of the concerned systems, we start with a simplified model of the environment, agents, actions, communications, and a required safety property to maintain by the agents. As an initial step, we simplified the context to grid-based autonomous multiagent systems, where the environment

is a grid and the agents present in the grid cells can sense the agents nearby and can have simple communication with them; the agents have their own goals and can autonomously plan and move towards their goals in the grid, update their plans, and try to reach their goals along with satisfying a simple safety property which is collision-avoidance with the others. Our initial plan is to find a way to automate testing of the agents in this simplified model, and then try to elaborate on the model (based on the scope of SafeSmart) step by step and provide solutions to more realistic domain systems.

1.3 Methodology

Random test case generation is a low-cost first step towards test automation. However, such random test cases may be less effective in detecting possible faults of the System Under Test (SUT). In our domain, each test execution is considerably resource and time-consuming, necessitating the effectiveness of random test cases even more. In this thesis, we aim to leverage the domain knowledge of test experts to guide the generation of test cases. In order to do that, first, we design a Domain Specific Language (DSL) to formalize the domain knowledge of test experts. Then, we generate test cases according to the guidance of the test experts formally specified in our DSL. If the test selection criterion suggested by the test experts is general enough, more than one concrete test case can be defined based on that criterion. In that case, we can generate different concrete test cases and pick some of them randomly for testing. Generating such test cases can be implemented in different ways. Generating random test cases and then filtering out the unwanted ones based on the domain experts' test selection criteria (formalized with the DSL) is one approach. Another approach is using constraint solvers for generating test cases directly based on the given criterion. We employ both of these approaches aiming for an efficient generation of test cases for our target systems, which will be discussed in detail later in this thesis.

1.4 Challenges and Research Questions

The challenges addressed by this thesis along with the corresponding research questions are presented in this section.

In the random generation of test cases, many test cases can be less effective. Using the domain knowledge of test experts, formally defined through a DSL, to filter out the potentially less effective test cases can lead to a more efficient fault detection process. The research question **RQ1** of this thesis concerns whether filtering out randomly generated test cases based on specific criteria

proposed by domain experts can improve the efficiency of the fault detection process in our target domain, as follows:

RQ1: *Can random generation and filtering test cases make fault detection more efficient in grid-based multiagent systems?*

The domain experts' criteria for filtering the random test cases can also be utilized in reducing the size of a failing test input. The concise failing inputs are useful for system analysts, for example in debugging and detecting the source of the fault in the system. To assess whether filtering random test cases improves the efficiency of reaching the most concise failing test input in our domain systems, the research question **RQ2** is posed in this thesis, as follows:

RQ2: *Can random generation and filtering test cases lead to a more efficient process for finding the most concise failing test case in grid-based multiagent systems?*

In the two research questions above, we used filtering randomly generated test cases to generate the test cases of concern. However, this could also be achieved by using constraint solvers [5; 6] to directly generate concerned test cases and pick random solutions from them. These two methods can have different efficiency in generating the concerned test cases. A comparison of the efficiency of the two approaches is addressed in the research question **RQ3** of this thesis as follows:

RQ3: *How does test case generation efficiency by random generation and filtering compare with test case generation by constraint solving in grid-based multiagent systems?*

The efficiency of the above-mentioned methods in the automatic generation of focused test cases can be affected by the size of different test setup parameters. The research question **RQ4** is posed to address how different test setup parameters affect the efficiency of these methods in generating test cases, as follows:

RQ4: *How do problem domain and constraint complexity influence test case generation time with either of the two methods in grid-based multiagent systems?*

To formally specify the test selection criteria, a simple DSL was designed concerning the research questions **RQ1**–**RQ4**. However, that DSL was limited in its features, and only simple test selection criteria could be specified with that. The research question **RQ5** of this thesis addresses how an extended version of that DSL can be introduced, as follows:

RQ5: *How to extend the DSL by generalizing its features to more focused test scenarios?*

While extending the initial DSL, some of its features were generalized and some features were added to it. It would be useful to evaluate if the DSL was extended in a useful manner and can specify interesting test scenarios in comparison with the initial DSL. The research question **RQ6** of the thesis addresses this concern, as follows:

RQ6: *Are the scenarios specifiable with the extended DSL more useful than the initial DSL ones?*

Using the extended DSL, generating more focused test cases is possible. Along with detecting the faults in different corner cases, the scenarios specified with the extended DSL can detect the faults more efficiently than the initial DSL. The research question **RQ7** is posed in this thesis addressing this issue, as follows:

RQ7: *Can the extended DSL be used for more efficient fault detection than the initial DSL?*

1.5 Contributions

Our contributions correspond to our research questions that were introduced in Section 1.4. In each paper included in this thesis, some of these research questions are addresses. We briefly mention the contributions of each paper in this section.

The contribution of Paper I is regarding the concerns of the research questions **RQ1** and **RQ2**. In this paper, first, a simple DSL is designed to formalize the specification of locality-based constraints. The constraints specified with this DSL are then used to filter out randomly generated test cases using the tool QuickCheck [7]. Then, an experiment is designed and conducted on a simple grid-based multiagent SUT, named *SafeTurtles* [8], with an injected fault. The introduced DSL is also implemented sufficiently to support the designed experiment. The experimental results are statistically analyzed and discussed at the end to answer the targeted research questions.

In Paper II, first, a more elaborate investigation for the same research questions **RQ1** and **RQ2** is provided. The experiment that was previously designed in Paper I is redesigned in Paper II. Some new faults are injected into the experimental SUT (SafeTurtles) and the experiment is conducted by more test setup parameter values. In addition to that, constraint solving is introduced as

an alternative method for generating the intended test cases. The DSL is fully implemented with QuickCheck/Erlang for filtering random inputs approach and with Z3/Python [6] for the constraint solving approach. The performance of these two methods and the effect of different test setup parameters on the performance of each method are discussed, which contribute to the concerns of the research questions **RQ3** and **RQ4**.

In Paper III, first, the previously proposed DSL for formalizing domain experts' test selection criteria is extended, which contributes to the issue raised in the research question **RQ5**. Then, to address the concern of the research question **RQ6**, an online survey [9–11] is designed and distributed among a group of test experts to collect their opinions about the scenarios that can be defined with the initial and extended DSL. The results of this survey are used to compare the usefulness of the initial and extended DSLs with each other. Furthermore, an experiment is designed and conducted, using SafeTurtles with a particular injection of faults, to compare the efficiency of fault detection with both DSLs, which addresses the concern of the research question **RQ7**.

Thesis outline: The rest of the thesis is organized as follows. The background and related work are presented in Chapter 2, and an overview of Papers I-III is discussed in Chapter 3. Finally, the conclusion and our plan for future work are presented in Chapter 4.

2. Background and Related Work

In this chapter we begin with a general description of our domain systems, i.e., grid-base multiagent systems. Then, we discuss the testing approaches for the concerned systems and the related work. At the end, we provide an overview of two main tools used in this thesis for conducting the designed experiments.

2.1 Domain Systems

Inspired by the work of Russel and Norvig [12], we narrowed down our scope in this thesis to quality assurance of multiagent systems acting in a grid environment. As represented in Fig. 2.1, our target agents can communicate with the environment and other agents for perception. Then, they use such perceptions for autonomous planning and acting (moving or performing stationary tasks) in the environment.

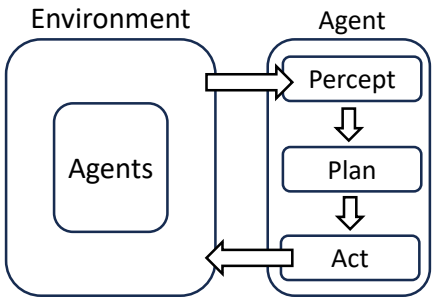


Figure 2.1: Domain systems

2.2 Related work

Different mechanisms are used in the literature for testing agent based systems [13]. In path traversal approaches [14–16], first, a traversable model of the SUT is prepared in the form of a state machine, Petri net [17] or AUML [18]. Then, different possible paths through that model are explored for generating

test cases. In information extraction approaches [19–21], the test cases are defined based on the information that is extracted from the system artifacts, such as design or ontology artifacts [22]. Random testing is another well-known approach that attempts to generate random inputs for testing [23; 24]. The random approach is also used to support other approaches by selecting numerical values in a required range [24]. In rule-based approaches, the test cases are generated by manipulating other test cases according to the defined rules, such as transformation rules or genetic operators [25; 26]. In this thesis, we provide a DSL to be employed by test experts and rely on their knowledge and analysis for specifying test cases. Therefore, we do not employ path traversal and information extraction approaches in this work. Using the DSL, a set of constraints can be specified to be used in test generation process, which is somehow similar to the rule-based approach. However, these constraints are supported by the random approach in our work for generating concrete test cases, which is different from the rules in rule-based approaches that describe how test cases are generated by modifying other test cases.

To tackle the effectiveness issues in random testing [1], several strategies have been proposed for enhancement. Adaptive Random Testing (ART) [27] emerges as a notable method, encompassing techniques such as filtering random data based on specific criteria or leveraging diversity in random data to improve fault detection rates. Our method for generating random movement plans for autonomous agents is based on our previous experiments, ensuring a level of diversity in the generated test inputs, which is in accordance with the methods outlined in ART. The integration of a formal diversity measure shows potential for boosting the effectiveness of random testing in our experiments, underscoring the necessity for further exploration and investigation.

Constraint solvers offer direct solutions for constraints, but employing them for testing and verification poses certain drawbacks and challenges. Firstly, scalability remains a concern, particularly when addressing large problems with intricate constraints, as this process can demand excessive time and resources. Secondly, when seeking a diverse set of test cases, it is often necessary to find varied solutions for a single constraint. This sampling of SAT solutions is commonly known as Constraint Random Verification (CRV) [5] within the hardware design domain and as SAT witnesses in other contexts. Smarch [28], Spur [29], UniGen2 [30; 31], and QuickSampler [32] represent some of the cutting-edge samplers. Among them, Spur specifically aiming to tackle scalability and uniformity concerns [33]. In this thesis, we employ the Z3 solver [6] for constraint solving and leverage its random seed to attain solutions for given constraints. We focused on investigating the time required for the solver to reach the initial solution, while analysis of solution diversity through repeated solver calls is deferred to the future research.

Random testing [27] and constraint solving [5; 6] explained above are placed in two opposite ends of one spectrum with respect to their computational requirements. Random testing, on the one end, expends minimal computational resources in generating test inputs but carries high uncertainty regarding the desired outcome, resulting in poor productivity in test selection. Conversely, constraint solving demands significant computational effort but offers assurance in capturing domain constraints, if feasible. Search-based approaches [34], positioned between these extremes, necessitate more computational effort than random input filtering yet less than full constraint solving. In our study, we scrutinize the efficiency of these two extremes to elucidate the performance characteristics crucial for selecting an appropriate approach. These characteristics can also be used in designing efficient search-based algorithms, potentially by integrating random filtering and constraint solving when neither approach individually demonstrates satisfactory performance. It is worth noting that search-based approaches are not entirely distinct from constraint solving, as contemporary SMT solvers utilize optimization engines to support the solving process. For instance, the optimization engine in Z3 can be directly accessed for generating test cases when the test specification is framed as an optimization problem. Apart from search-based testing, there are other approaches that enhance the performance and effectiveness of both random and constraint-based testing. While our study focuses on baseline approaches, these enhancements, briefly mentioned above, could be explored in future empirical investigations.

Scenario-based testing is a prevalent approach for evaluating autonomous vehicles. Notably, organizations such as ASAM [35], EuroNcap[1], and DOT [36] have formulated scenarios and specification languages tailored for this purpose. These test scenarios can be derived from crash data analysis [37; 38] or naturalistic driving data [39–41], with each scenario typically focusing on a specific corner or critical case of the system. Testing various configurations of critical situations can enhance confidence; however, conducting real-world tests for numerous cases proves impractical due to its costs, test setup and execution challenges and limitations, and safety concerns. Simulation environments such as SVL[2], OpenDS [42], SUMO [43], and Gazebo [44] offer safer and more efficient alternatives for executing tests on autonomous systems, and they would reduce the need for conducting operational tests. However, conducting some tests in the operational environment may still be necessary to gain acceptable level of assurance of the target systems. Simulation environments facilitate automating the testing process for autonomous agents, and as a result, we prefer to employ them in our work. Therefore, in designing test

[1]https://www.euroncap.com/en

[2]https://www.svlsimulator.com

suites, we do not consider the test setup and execution challenges of the scenarios that may be of concern in operational environments. Instead, our focus in this thesis is to automatically generate different corner cases of intended critical situations that would be easily executed later on in simulation environments to detect the potential faults of the system efficiently.

There are a number of domain specific languages in the literature for describing scenarios for testing cyber physical systems and automated vehicles (AV) such as OpenSCENARIO [35], GeoScenario [45], SCENIC [46], and M-DSL[1]. These languages are designed based on particular goals and each have its own strenghts and weak points [47]. In contrast to other DSLs, we initially chose to develop a formally-defined and minimalist DSL that concentrates on locality constraints within multiagent grid-based systems. Our design strategy aimed to offer a focused DSL, enabling meticulous exploration of parameter effects on the efficiency of test-case selection mechanisms. While future investigations may be required to validate our findings with more intricate DSLs, our outcomes will serve as a blueprint for organizing such studies. Later on, we extended the DSL, to cover specifying more elaborate scenarios and also conducting further research on the efficiency of test scenarios. Although it is possible to specify concrete test scenarios with our extended DSL, unlike many other languages, its goal is to specify just the characteristics of the considered critical scenarios thus offering more flexibility and variation in the generated test cases. Such specification can be prepared faster than concrete scenarios with all details. Moreover, it allows employing randomness to generate different corner cases of one critical situation automatically.

2.3 Tool Overview

In this work, we utilize QuickCheck and Z3 as the tools for implementing our approach and conducting the necessary experiments. This section provides a brief introduction to them.

2.3.1 QuickCheck

Within the framework of the SafeSmart project, we employ QuickCheck[2] [7], an advanced Property-Based Testing (PBT) tool. QuickCheck facilitates automatic input data generation through specialized random data generators tailored for various data types, including numbers, lists, and vectors. Additionally, it offers the capability to combine generators to construct more intri-

[1] https://www.foretellix.com/open-language

[2] `http://www.quviq.com/products`

cate data structures. QuickCheck enables the selection of specific test cases by filtering automatically generated test inputs using a predefined predicate. QuickCheck is developed in Erlang [48] programming language and integrated in it. Thus, its users use this language to model and generate test cases with the tool. Erlang is a functional, weakly typed, inherently distributed, and platform-independent language.

In QuickCheck, when a generated test fails, the tool initiates a process known as *shrinking* to simplify the analysis of the failed scenario and debugging. During shrinking, QuickCheck endeavors to discover a more concise failing test input. After defining a more concise candidate, QuickCheck re-executes the SUT with that. If the test fails with that input, QuickCheck moves closer to identifying one of the most concise failing inputs, marking this as a *successful shrinking* attempt. Conversely, if that input does not result in a failed test, QuickCheck backtracks and explores alternative methods of reducing the test input, marking that as a *failed shrinking* attempt. This iterative process continues until no further successful shrinking attempts are possible, at which point the last input is identified as the most concise failing test. In the case of having a test selection filter, QuickCheck utilizes the same filter during the shrinking process. By default, QuickCheck assumes that inputs not meeting the filtering criterion would not induce test failure. Therefore, if a more concise input violates the filtering constraint, it is simply discarded. We use QuickCheck in this thesis to generate randomized test cases with and without test selection filters. The test report of the QuickCheck includes several details, including the number test execution attempts and the number of successful and failed shrink steps, that will be used in this work for conducting statistical analysis and comparing the efficiency of different approaches with each other.

2.3.2 Z3 SMT solver

Z3 [6], developed by Microsoft Research, is a state-of-the-art constraint solving technology. It belongs to the Satisfiability Modulo Theory (SMT) solver family, enabling the expansion of Boolean satisfiability checking by incorporating predicates from various theories beyond Booleans, including integers, sequences/arrays, and more complex data types. Implemented in C++, Z3 offers APIs for multiple programming languages such as Java and Python. Like other constraint solvers, Z3 takes input comprising intended variables, their domains, and the constraints linking them. It then endeavors to find a set of assignments to all variables within their domains that satisfy the provided constraints, presenting this as a solution. One noteworthy feature of Z3 is its ability to diversify produced solutions using a random seed, which proved beneficial

in our application. In this thesis, we use Z3 for generating different test cases based on a test selection criterion, and its performance is compared with our other approach. There are many other constraint solvers available that could be used as an alternative to Z3, such as Minizinc[1], Choco[2], and Gecode[3]. We chose Z3 in this work due to its popularity and being well-known for its high performance and having very flexible and multi-language APIs.

[1]https://www.minizinc.org/
[2]https://choco-solver.org/
[3]https://www.gecode.org/

3. Summary of Appended Papers

3.1 Paper I

The target of Paper I is automating the testing process of multiagent systems that move autonomously in a grid environment. In the aimed systems, each agent begins from a starting point and tries to reach its goal position in the grid. To reach their goals, each agent takes a suggested path to follow at the beginning. It is recommended for the agents to follow their suggested paths, but they can update their plans and follow another path when needed. In this journey, it is also expected from each agent to move safely by considering the other agents and avoiding collisions with them. Therefore, in critical situations when following their planned paths can potentially lead to a collision, the agents are expected to update their plans accordingly and execute it on time to maintain the safety.

Our aim of testing the concerned systems is to evaluate if the agents are moving safely and the test oracle, i.e., not having collisions among agents, is always satisfied, regardless of how critical the agents' situation is. Test automation for these systems can be done by generating random test cases. However, many of those random cases can take the agents towards situations that the test oracle violation is not even possible, which leads to a less effective testing process in terms of quick fault finding. Since each test execution is significantly resource and time consuming in our domain, execution of all such random test cases can be very costly too. Filtering out less effective random test cases can reduce the number of test executions and improve the efficiency of this approach. In order to do filtering, a filtering criterion is required. But how should such a criterion be defined? For each system, such filtering criteria can be defined by the domain experts of that system based on their knowledge about the critical situations and potentially effective test cases of that system. The domain experts can provide their criteria of effective scenarios simply in natural language format. However, to avoid ambiguity and facilitating automatic employment of those criteria in test automation, we advocate devising a DSL for formalizing the domain knowledge.

To have a formalism of domain knowledge with respect to the multiagent systems, we introduce a simple DSL in this paper for specifying locality-based

test selection criteria, with the grammar syntax shown in Mod 3.1. This DSL allows test experts to specify particular constraints based on the location of the agents at different times. As an example, one criterion specified by this DSL can require having five random agents that plan to get closer to each other and all visit a square with side lengths one unit at the same (but not decided upfront) time. This constraint is specified with our DSL as "`In Square 1 Count 5`". To generate test cases satisfying such a constraint, we can simply generate random inputs, which are the initially recommended paths of the agents in our case, and just filter out the ones that do not satisfy this criterion.

Module 3.1: DSL syntax for specifying locality-based test selection constraints

```
1  Constraint ->   In  Area Condition |
2                  And Constraint Constraint |
3                  Not Constraint |
4                  Or  Constraint Constraint
5
6  Area       ->   Circle Integer |
7                  Square Integer
8
9  Condition  ->   Count Integer |
10                 Intersection Integer Integer |
11                 And Condition Condition |
12                 Not Condition |
13                 Or  Condition Condition |
```

In comparison with just randomly generated test cases, we anticipate a higher efficiency in terms of fault finding by applying the filters. To investigate this issue, we design and conduct an experiment in this paper. This experiment is based on a simple case study of autonomous agents that move in a grid environment, which we implemented and called it *SafeTurtles* [8]. The decision making algorithm of these agents is implemented properly, and they can avoid collisions with the other ones in critical situations. However, for experimental purpose, a fault is injected to them, making them susceptible to wrong moves and collisions with the others. In our experiment, we try to detect the fault of the system with both randomly generated test cases and filtered random test cases with a few filtering criteria. We employ QuickCheck [7] tool for generating test cases of both methods and apply two measures in this experiment. First, we count the number of test executions in each test until detecting the fault of our SUT (SafeTurtles). Since test execution is very resource and time consuming in our domain, we take that as a criterion to compare the efficiency of testing by each method. Second, we consider the number of test executions to get one of the most concise failing test inputs, which is a valuable input in

debugging and analyzing the nature of the failure and localizing its origin in the source code.

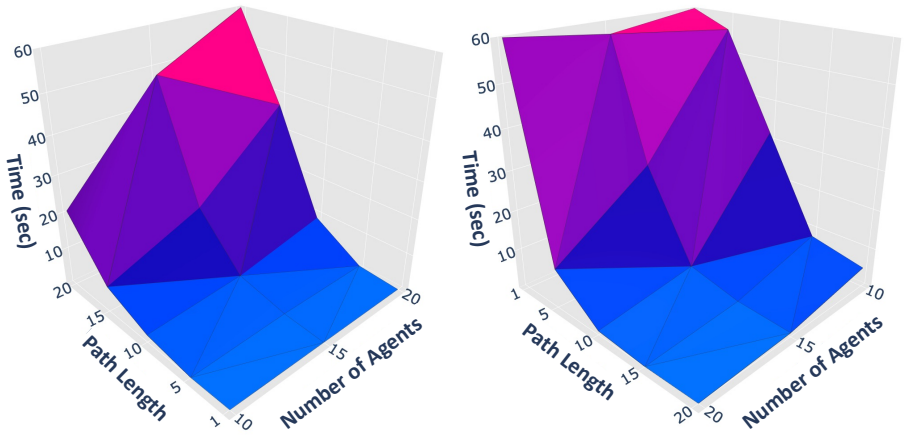
By running the experiment and conducting statistical analysis on the experiment results, we noticed that the number of test executions until detecting the SUT fault in each case of having a filtering criterion is significantly less than the case of having no filter. In our domain, it means that applying filters significantly improves the efficiency of testing with randomly generated test cases. In addition, our analysis showed that in the process of shrinking the size of failed test inputs, the number of test executions that lead to unsuccessful failed shrink attempts (by QuickCheck tool) is significantly reduced by having filters. It means that we can reach one of the most concise failing inputs more efficiently by having filters on random test cases.

3.2 Paper II

The work of Paper II can be split into two parts. In the first part, we try to address some threats to the validity of Paper I and elaborate on the experiment that we designed and conducted in that paper. In the second part of this paper, we use constraint solving as an alternative method to filter random test cases and generate the intended test cases directly from the domain experts' criteria.

We elaborate the work of Paper I with a few expansions. First, we injected multiple faults to our experimental SUT. To define which fault to inject, we investigated the common faults of autonomous robots designed in a project akin our experimental SUT by interviewing two lecturers of the graduate course *Design of Embedded and Intelligent Systems (DEIS)* at Halmstad University [49]. Second, we add one more criterion to measuring the efficiency of fault detection in our experiment. We count the number of execution steps in the SUT till detecting a fault in each test execution in the new version of our experiment. In other words, we measure how long each test needs to run until failure detection. This criterion provides a more precise indication of either the necessary timing cost or the size of the test suite needed to identify faults in our SUT. Third, we extend the experiment execution to the environments with different grid sizes to have more confidence in the experimental results at the end.

Running the newly elaborated experiment and applying statistical analysis on the results confirm our findings from Paper I. Our statistical analysis shows that applying test selection filters significantly reduces the number of test executions and the number of steps in each execution in different test setups. It also shows that the number of failed shrink steps in several test setups is significantly reduced by applying filters to random test cases, which results in having more efficient process in reaching one of the most concise failed test inputs.



(a) Constrain Solving **(b)** Filtering random inputs

Figure 3.1: The time of generating one test case (on average for 30 runs) satisfying the constraint "`In Square 1 Intersection 1 5`"

In Paper II, along with filtering random inputs, we use constraint solving method to directly generate the test cases that satisfy the criteria proposed by domain experts through our DSL. These approaches show different performances for different test selection criteria and different test setups. As an example, figure 3.1 shows the performance of these methods to generate one test case based on one test selection constraint. Note that in this figure, the sub-figures 3.1a and 3.1b are presented intentionally from different point of views (by rotating their axes) to demonstrate their details in a clearer way. In this paper, we aim to find out which method is more efficient to use for generating test cases in different situations. To investigate this issue, we design an experiment to generate test cases for different test setup parameter values and scenario constraints and measure the time of test case generation by both methods.

In this experiment, we have four test setup parameters: grid size, the number of agents, the path length of the agent, and the test selection constraint. By running the experiment and conducting statistical analysis on the results, we noticed a complementary performance of these two approaches based on the problem parameters. Based on our observations and analysis, the value of grid size and constraint strictness directly impact the test generation time of filtered random inputs approach. However, the performance of the constraint solving approach seems robust to these parameters and does not change significantly by them. Conversely, the number of agents and path length directly impact the time of test generation with constraint solving approach, although these fac-

tors do not have a significant effect on the test generation time of the filtered random inputs.

3.3 Paper III

By exploying the DSL that we introduced in Paper I (shown in Mod 3.1), we could only specify particular constraints related to the location of the agents at some times in the grid. In Paper III, we extended this DSL to allow test experts to specify more realistic test scenario constraints. To do that, we first upgrade our target SUT and our model of the environment and the agents, as shown in Fig 3.2. In this work, we consider a dynamic environment modeled as a collection of grid cells, where each grid cell and agent can have tailored attributes. Each defined attribute is associated with a value at a time during SUT execution, where the set of all attributes of an entity (agent or grid cell) and the values associated with them at a time defines the *state* of that entity at that time. Each entity can have a plan in our model to reconfigure itself to a particular state at a time. To run our SUT, defining the plans of all entities at all times is needed at the beginning. For testing, the test oracle property should also be defined along with the SUT inputs. By running the system and monitoring the states of the agents during test execution, we can check if the given test oracle property is satisfied by the agents or not.

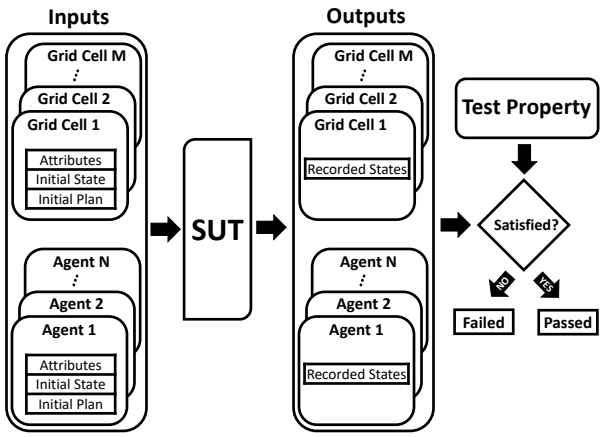


Figure 3.2: The target SUT with its inputs and outputs

For the modeled systems, we propose a DSL to specify test selection constraints based on the planned state of different entities at different times. We have more flexibility in defining the test oracle using this DSL too. For example, rather than just forbidding collisions, we can test if the agents are not

getting closer than a particular distance, or they do not enter particular zones in the environment by the new DSL. To be able to specify more focused test scenarios using this DSL, we generalize the features of the locality-based elements that we introduced in the DSL of Paper I. As a result, we can generate test scenarios that can guide the systems towards more focused critical situations. Furthermore, for the faults that both the initial and the extended DSL can target by their own test scenarios, the extended DSL is anticipated to show a better efficiency than the initial one. To investigate this issue, we design an experiment with few test scenarios specified with the initial DSL and then design similar but more focused version of such scenarios with the extended DSL. We conduct our experiment on SafeTurtles [8] which we introduced in Paper I, however, we inject the faults to it differently in this experiment (the faults are injected when the agents are in a particular zone of the grid). We count the number of test executions, the most costly task in this context, until detecting a fault of the SUT using the set of automatically generated test cases according to the scenarios specified with both of our DSLs. Running the experiment and conducting statistical analysis confirmed our anticipation of the performances. The results show that the test scenarios defined by the extended DSL can significantly reduce the required number of test executions to detect the system fault, which results in a higher efficiency of the extended DSL scenarios.

In Paper III, we also concerned if the DSL is extended with an added value from the point of view of test engineers. To address this issue, we designed an online survey to collect the opinions of different test experts about the initial and extended DSL. The questionnaire includes two scenario constraints that are specifiable with the initial DSL. A more focused version of those scenarios specifiable with the extended DSL are also included in the survey. In addition, three other kinds of scenario constraints are added to the survey which cannot be addressed by the initial DSL, but they are specifiable with the extended DSL using the extended features. By spreading the questionnaire and collecting the responses, we could see different rankings for each scenario by different respondents. According the survey results, the test experts find the kind of scenario constraints that are specifiable with the initial DSL more useful than the kind of scenario constraints that can be specifiable with the newly added features of the extended DSL. However, among the kind of constraints specifiable with the initial DSL, the respondents find the more focused version of such constraints more useful, which are specifiable with the extended DSL.

4. Conclusions and Future Work

This thesis aims to efficiently automate the testing of autonomous grid-based multiagent systems. This purpose is initially pursued in this thesis by employing domain knowledge of test experts and random testing approach. To avoid having ambiguity in specifying the domain knowledge of test experts, a basic DSL is devised to provide domain experts a formalism to specify particular locality-based test selection constraints. Then, the efficiency of utilizing this DSL to filter out less effective random test cases is compared with purely random test cases. This comparison is conducted by designing and running an experiment based on a simple system of grid-based multiagent systems (SafeTurtles) with an intentionally injected fault. By statistically analyzing the results of this experiment, it is shown that utilizing the domain knowledge of test experts can significantly reduce the number of test executions for detecting faults in the system. Having test execution as the most resource and time-consuming task of testing in our domain, a significant reduction of the number of test executions will lead to significant improvement of the fault detection process. Furthermore, the statistical analysis of the experiment results revealed that applying the domain knowledge significantly improves the process of reaching the most concise failing test input too, which is a valuable input in debugging and analyzing the faulty systems and facilitates locating the source of the found faults.

Instead of filtering randomly generated test cases, constraint solvers can be applied to the domain knowledge of test experts to directly generate test cases. To find out which method is more suited for generating particular test cases, the performances of these methods are compared with each other in this thesis. In an experiment, both of these methods are used to generate test cases for different test setup parameters. The experimental results show that test setup parameters affect the performance of these approaches differently. In particular, these approaches show complementary strength based on the experimental results. Filtering randomly generated test inputs has a better performance when the number of agents is large and the agents' paths are long, while larger grid sizes and more strict constraints degrade its performance. On the other hand, constraint solving shows a robust performance for large grid sizes and strict constraints, while more agents and long paths deteriorate its performance.

The first proposed DSL for formalizing the domain knowledge of test ex-

perts was limited in its features. Therefore, the DSL is extended in this thesis to be able to formally specify more elaborate domain knowledge. In order to do that, inspired by the work of Russel and Norvig [12], the model of the agents and environment is improved. Then, the DSL is extended by adding some features based on the new set of inputs and outputs of the elaborated model. In the extended DSL, along with specifying the agents' behavior in the test cases, a dynamic environment can also be specified. Different attributes can be defined for the agents and the environment in the DSL, and complex test scenarios can be specified based on them.

The extended DSL is compared with the initial one in the thesis from two perspectives: performance and test experts' interest. To collect the perspective of the test experts, an online questionnaire is conducted. In this survey, some test scenarios in natural language were proposed to the respondents. Some of these scenarios can be specified with the initial DSL. The other ones can only be specified with the extended DSL that consists of i) more focused version of the same survey scenarios specifiable with the initial DSL ii) the scenarios of the kinds that are not addressed by the initial DSL. The survey results show that the new kinds of constraints in the extended DSL are found less useful but the more focused version of the same initial kind of constraints in the extended DSL are found more useful by the test experts than the constraints specifiable with the initial DSL. In addition, to compare the efficiency of the DSLs, the extended DSL is sufficiently implemented to conduct a designed experiment. The results of the experiment showed that the test cases generated by the extended DSL detect the SUT faults with a significantly smaller number of test executions, leading to a more efficient test process.

As our next step, we are considering testing a more realistic system in our domain by utilizing our extended DSL. We would like to use an autonomous robot and test it with its newly implemented features before putting that on the product line. Testing open-source systems of autonomous agents like Appolo[1] is another option. For any system that we decide to test, we plan to interview the chosen system's test experts about the potentially useful test scenarios. Analyzing such test scenarios may lead to further extension of our DSL to be able to express the exact required scenarios. And finally, we aim to have a full implementation of our DSL at the end to generate the test case automatically. For implementing the DSL, we will consider all of the lessons that we learned from experimenting with the performance of constraint solving and random input filtering approaches. We hope we can find a way to combine these approaches in the implementation so that the test cases are generated efficiently for the required scenario constraints and test setup [27; 34]. More specifically,

[1] https://github.com/ApolloAuto/apollo

we will consider the following research questions in our future work:

- Which test scenarios are suggested by the test experts for a particular realistic grid-based multiagent system?
- How our DSL can be upgraded to specify the suggested scenarios of the test experts for testing the particular grid-based multiagent system?
- How an efficient search-based technique can be devised by combining different approaches to be used in the implementation of the DSL and generate the required test cases with a satisfactory performance?

Sina Entekhabi

"Sina Entekhabi received his M.Sc. degree in Computer Engineering from Middle East Technical University (METU), Turkey, in 2018. Then, he worked shortly at Chalmers University, Sweden, as a project assistant, and, soon after, he joined the School of Information Technology at Halmstad University as a Ph.D. student in September 2019. In his doctoral studies, he focuses on automating the testing process for autonomous systems. In January 2023, Sina joined Phoniro (ASSAABLOY) company as a Quality Assurance (QA) engineer."

School of Information Technology

ISBN: 978-91-89587-49-6 (printed)
Halmstad University Dissertations, 2024

HALMSTAD
UNIVERSITY