Learning in Uppaal for Test Case Generation for Cyber-Physical Systems

Rong Gu[0000−0003−0570−6005]

Mälardalen University, Sweden
`rong.gu@mdu.se`

Abstract. We propose a test-case generation method for testing cyber-physical systems by using learning and statistical model checking. We use timed game automata for modelling. Different from other studies, we construct the model from the environment's perspective. After building the model, we synthesize policies for different kinds of environments by using reinforcement learning in Uppaal and parse the policies for test-case generation. Statistical model checking enables us to analyse the test cases for finding the ones that are more likely to detect bugs.

1 Introduction

Cyber-physical systems (CPS) are becoming pervasive in modern society. Such systems are not pure software or hardware but consist of cyber components (i.e., software controllers) and physical components. With the development of artificial intelligence, autonomous systems, a recent example of CPS, are becoming more and more realistic. Such systems, e.g., self-driving cars, run autonomously by perceiving the environment via sensors, making decisions via controlling software and interacting with the environment, such as moving and carrying goods. CPS are often designed to accomplish specific tasks that are repetitive and tedious for humans. On some occasions, CPS have to work alongside humans, such as on construction sites. In this case, the safety guarantee of CPS are crucial as a subtle fault in the system can lead to casualties.

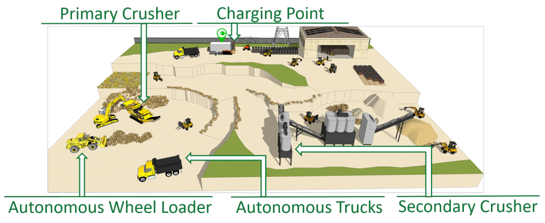

Fig. 1. An example of CPS: an autonomous quarry

Figure 1 depicts an example of CPS working in an autonomous quarry. The quarry contains various autonomous CPS such as trucks and wheel loaders. In this example, the mission for the CPS are transporting stones in a quarry, where wheel loaders dig and load stones, and trucks transport stones. First, the wheel loaders need to move to stone piles, dig stones, and load them into trucks. Then, the trucks carry on to transport the stones to the primary crushers, where stones are crushed into coarse fractions before they are carried to the secondary crushers. When working in the quarry, the CPS must transport a certain amount of stones within 24h to keep a high level of productivity. They also must avoid obstacles such as rocks and other machines and visit the charging point periodically. To achieve the goal, the CPS must be able to find safe trajectories, track the trajectories closely to avoid obstacles and be ready to handle unexpected situations, such as animals suddenly appearing in the field.

When testing such systems, knowing whether the control software is functioning correctly is not enough. For example, when a wheel loader is putting stones in a truck, even if its controller sends the correct signals to the arm and bucket, the collaboration may still fail because the truck, which is part of the environment from the wheel loader's perspective, may be parking at the wrong position or turning to a wrong orientation. Therefore, testing must consider not only the system itself but also its working environment. Additionally, the wheel loader must be unloading stones preciously at the right moment when the truck's position is under its bucket. Hence, the correctness of the system depends on its own functions, the working environment, and the temporal order and timing of performing actions.

Another factor that makes the problem even more difficult is that the environment can be uncertain. In the example of the autonomous quarry, all machines are collaborating, so we can assume the environment to be friendly, which means every system is working toward the same goal. However, in some situations, the environment is neutral or even antagonistic. For example, from a self-driving car's perspective, pedestrians and other drivers are hard to predict. Some of them are friendly but some of them are aggressive, e.g., suddenly turning their direction without using the indicator. To test CPS working in such an environment, we need to cover all the possible situations so that the system is ready for emergencies. However, the environment is hard to model, and some even contain rare events that are very unlikely to happen but once they appear, accidents occur. In summary, testing such systems is extremely difficult and we need to consider not only the system but also the environment.

In this paper, we propose a method that uses reinforcement learning [9] to train various traffic agents such that the combination of their behaviour can form different types of environments. Based on the design of the reward function for reinforcement learning and the primitive behaviours of the environment, we can train the environment to behave in different ways, and the training is carried out in Uppaal Stratego[1] [4]. The tool enables us to not only perform the training but also verify the results by using its statistical model checker (SMC) [3]. The trained environment provides the test cases for the CPS and we use the quantitative answers of verification to evaluate the quality of the generated test cases. Most importantly, as the modelling, training, and verification are all done by using formal methods, the unreliable results of reinforcement learning are now strengthened by the rigour of formal methods, and the automatically generated test cases can be used for finding sophisticated bugs, like the ones concerning

[1] Uppaal Stratego is now integrated into Uppaal 5: uppaal.org.

the temporal order of task execution and the timing constraints, which is not achievable by other automatic testing techniques, such as fuzzing [8].

2 Test-Case Generation by Learning in Uppaal

In this section, we introduce the method for the generation of test cases for cyber-physical systems. In this method, we use the modelling language *timed game automata* to build the model of CPS and environment. Timed game automata (TG) are special timed automata (TA). The latter is finite-state automata extended with real-valued variables that increase at the same rate one [1]. TG further extends TA by partitioning actions into *controllable* and *uncontrollable* ones.

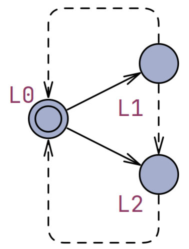

Fig. 2. TG example

Figure 2 depicts an example of TG, in which solid arrows represent controllable actions of the system and dotted arrows are uncontrollable actions of the environment. In Uppaal, circles are called *locations* and arrows are called *edges*, whose formal definitions are in the literature [4]. However, readers are not required to understand these concepts for reading this paper. Intuitively, when the TG example is at location `L0`, it has three options of controllable actions. Two are explicit, that is, going to location `L1` and going to location `L2`. The third option is waiting at `L0` until an uncontrollable action takes place, which is implicit as it is shown in the automaton. In Fig. 2, since there is no uncontrollable action at location `L0`, to choose to wait there means to stay at `L0` for an unbounded time. At location `L1`, the uncontrollable actions may lead the model back to `L0` or further to `L2`, whereas at location `L2`, there is only one uncontrollable action getting back to `L0`. Policy synthesis means calculating a set of state-action pairs that shows the TG which controllable actions to choose at each of the states such that the model satisfies some properties, e.g., eventually coming back to the initial location `L0` within two steps or five time units.

TG have been applied in many real-world case studies [5][2][7]. In these studies, CPS are modelled as TG where controllable actions belong to the system and uncontrollable actions belong to the environment. Then they use reinforcement learning in Uppaal to synthesize policies for guiding the controllable actions of the system and analyse the results by using SMC or exhaustive model checking. In this paper, we construct the model in the opposite way. We model the environment's behaviour as controllable actions and the system's behaviour as uncontrollable actions. Then we use reinforcement learning to synthesize policies for the environment such that it knows how to *win* the game. If we switch the controllable and uncontrollable actions in the TG example and train an *unfriendly* environment, the resulting policy could tell the environment to go to location `L2` when the model is at `L1`, which makes the system lose the game, i.e., coming back to `L0` within two steps.

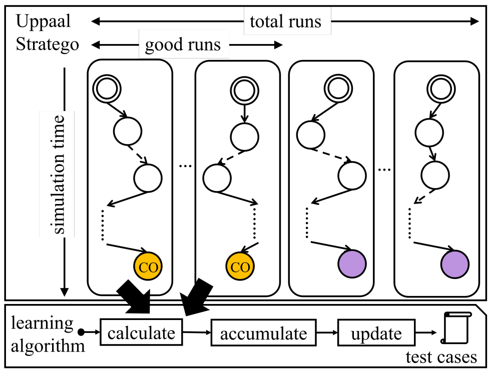

Fig. 3. The process of test-case generation, adapted from the literature [6].

We proposed a method for generating test cases by parsing the policies learned in Uppaal previously [6]. In this paper, we leverage the tool for test-case generation. Figure 3 shows the process of the method. First, we randomly simulate the model for a number of episodes and gather the runs of the model. Some of them are *good runs* that satisfy our property (i.e., environment winning the game) and some of them are *bad runs* that are abandoned for learning. After a certain amount of learning episodes, the policy becomes stable and we generate test cases, which represent the environment's behaviour. Further, we can use SMC in Uppaal to analyse those test cases for finding the ones that are more likely to detect bugs in the CPS.

In summary, our method is able to train different kinds of environments, which is crucial for testing CPS. Although the training is via reinforcement learning, because of the formal techniques in our method, we can generate test cases that are proven to be more likely to detect bugs in the systems, and the bugs can be sophisticated such as temporal-logic-based ones.

References

1. Alur, R., Dill, D.: Automata for Modeling Real-time Systems. In: Automata, languages and programming. Springer (1990)
2. Dai, S., Hong, M., Guo, B.: Synthesizing power management strategies for wireless sensor networks with uppaal-stratego. International Journal of Distributed Sensor Networks (2017)
3. David, A., Du, D., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. arXiv (2012)
4. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: UPPAAL stratego. In: TACAS 2015: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer (2015)
5. Eriksen, A.B., Huang, C., Kildebogaard, J., Lahrmann, H., Larsen, K.G., Muniz, M., Taankvist, J.H.: Uppaal stratego for intelligent traffic lights. In: 12th ITS European Congress (2017)
6. Gu, R., Enoiu, E.: Model-based policy synthesis and test-case generation for autonomous systems. In: 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE (2023)
7. Gu, R., Seceleanu, C., Enoiu, E., Lundqvist, K.: Model checking collision avoidance of nonlinear autonomous vehicles. In: Formal Methods: 24th International Symposium, FM 2021. Springer (2021)
8. Li, J., Zhao, B., Zhang, C.: Fuzzing: a survey. Cybersecurity (2018)
9. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)