ISBN: 978-981-18-5852-9
WCSE 2022 Spring Event: 2022 9th International Conference on Industrial Engineering and Applications
doi: 10.18178/wcse.2022.04.140

Safe Reinforcement Learning through Hierarchical Shielding with Self-Adaptive Techniques

Prasanth Senthilvelan [1], Jialong Li [2] and Kenji Tei [3]

[1] Department of Computer Science and Communication Engineering Waseda University Tokyo, Japan

[2] Department of Computer Science and Communication Engineering Waseda University Tokyo, Japan

[3] Department of Computer Science and Communication Engineering Waseda University/ National Institute of Informatics Tokyo, Japan

Abstract. This paper introduces a method for safe reinforcement learning that combines our hierarchical shielding approach with self-adaptive approaches and can be applied to AI systems or machines in the industries that use Reinforcement Learning (RL) while ensuring safety. A shield is a small monitor constructed from safety specification that is placed behind the RL agent. It monitors the environment and the agent's actions, and if the agent tries to do something dangerous to the environment, the shield overwrites the dangerous action with a safer one. Hierarchical shielding, on the other hand, has several levels of safety requirements, as well as different levels of shield that will be selected and changed out over the run time. We use graceful degradation and progressive enhancement to accomplish hierarchical shielding using self-adaptive approaches, such as the MAPE-K loop. We demonstrate it in this paper using a hot water storage tank as an example.

Keywords: safe reinforcement learning, shield, mape-k loop, graceful degradation, progressive enhancement

1. Introduction

Reinforcement learning is becoming increasingly essential for organizations that deal with huge, complex problem areas regularly. They are based on reinforcement learning, which teaches systems how to behave by interacting with their surroundings[1]. In reinforcement learning[11], a software agent is taught to act independently in an unfamiliar environment. To train the agents, RL algorithms are utilized, in which the agent learns a policy that maximizes a final reward by performing trial and error on the environment and collecting rewards. Even if RL algorithms find policies that optimize reward, they do not always ensure safety during the learning or implementation stages. In RL, the concept of trial and error is used, with the error being an important component of teaching the agent how to behave best. In the actual world, trial and error[6] can result in injury or even death. Furthermore, existing state-of-the-art RL approaches are insufficient for safely learning optimal control policies. To overcome the limitations of the current research technique for safe reinforcement[14] learning in the shielding approach[2] synthesizes safety standards and acts as enforcers[3]. If a violation is going to occur, they prevent any activities proposed by the agent from being carried out in the environment. But the problem with the existing single shielded approach is that for a continuously changing environment, one static shield is not enough. That is because in some situations we can never guarantee safety for sure. After all, other objects might cause damage to the agent, or sudden changes in the environment would be a cause of some danger. The existing single shield approach cannot change the safety specification to a risk-taking specification because the existing shield does not accept risk and the shield does not allow the agent to take any unsafe action and they guarantee us that no matter what happens there will never be something unsafe. For example, industry application requires safety and any malfunction of that machine can cause huge damage to everyone. So, there is a need for the system to degrade its safety specification and maintain its control. So, there is a need for a shield that can accept risk and a shield of different levels from where it can change during the run time. We want the system to improve the shield when there is an environment change. Since the shield is generated from safety requirements, the agent should change the safety requirement when there is an environment change. So that is why we propose our hierarchical shielding method with a self-adaptive technique to overcome this problem.

The remainder of this paper is organized as follows: Section II describes the motivating example, a hot water storage tank. Section III gives an overview of the problem with the existing shielding approach. Section IV introduces the proposed hierarchical shielding approach. In Section V, we evaluate the proposed method in a hot water storage tank scenario. Section VI is the discussion on how our proposed method is better than the existing method. Section VII introduces similar research. Section VIII gives the conclusion.

2. Motivating Example

Consider an energy-efficient controller for a hot water storage tank as an example shown in Fig. 1. A heater keeps the water in the tank warm, and its energy consumption is determined by the tank's filling level. When the valve is open, the outflow is between 0 and 1 liters per second and the inflow is between 1 and 2 liters per second (and 0 otherwise). The tank has a capacity of 100 liters, and whenever the inflow is turned on or off, the setting must be maintained for at least three seconds to avoid valve wear. Also, the tank must never overflow or run dry, regardless of whether it is at level 0 or level 100. If the valve is opened and then closed, it must remain closed for two additional time steps (seconds), and vice versa. When the water level in the tank drops too low or rises too high, the shield will enforce.

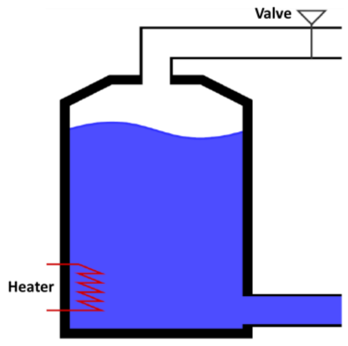

Fig. 1: A hot water storage tank.

3. Problem

In the classic reinforcement learning scenario, their approach introduces a shield. The shield is calculated ahead of time using the safety part of the system specification and an abstraction of the agent's environment dynamics. It assures safety and minimal interference, which means that the shield restricts the agent as little as possible and only prevents acts that could jeopardize the safe operation of the system. System specifications are given as temporal logic. The safety specification is transformed into an automaton, with only safe states available for exploration. The environment abstraction is then transformed into an automaton, which is frequently modeled as MDP[17]. The shield is then utilized to enforce the safety specification by solving a safety game based on the safety specification and environment abstraction, which is won if the system only visits safe states[2]. But the problem with the existing single shielded approach is that for a continuously changing environment, one static shield is not enough. That is because in some situations we can never guarantee safety for sure for example other objects or sudden changes in the environment might cause damage to the agent. There will be a need for a change in the specification or risk-taking specification to overcome that situation. One static shield would not take any risk. The existing single shield approach cannot change the safety specification to a risk-taking specification because the existing shield does not accept risk. So, there is a need for a shield that can accept risk and a shield of different levels from where it can change during the run time. We want the system to improve or change the safety requirement when there is an environment change. So that is why we propose the hierarchical shielding method with self-adaptive technique to overcome this problem.

4. Hierarchical Shielding for Safe Reinforcement Learning

4.1. The main architecture

The main architecture of our proposal as shown in Fig. 2, contains two parts which is the managed system and the other which is on top of it is the adaptation engine. The managed system consists of environment, policy, reward function, and shield, which is a typical architecture of shielded reinforcement learning. The adaptation engine consists of Monitor, Analyze, Plan, Execute with the Knowledge component. In the shielded reinforcement learning run time architecture that is in the managed system, the state of the environment goes from the environment to the policy where the policy gives the action to the shield and the shield gives a safe action which is the modified action to the environment. Then from the environment, it goes to the reward function and gives the reward to the policy. But for our proposal, we attach the adaptation engine[15] on top of the managed system.

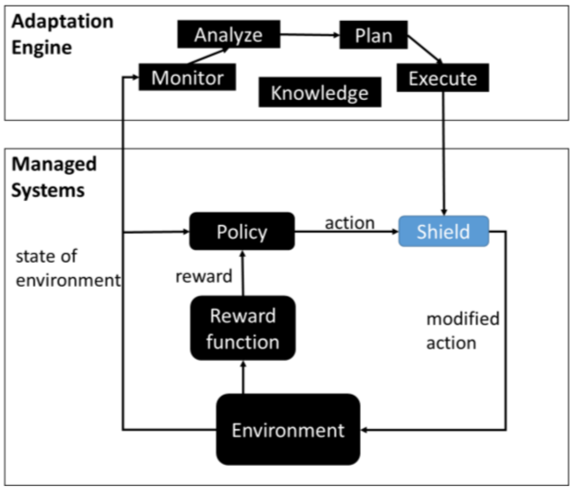

Fig. 2: Hierarchical Shielding Architecture with MAPE-K loop.

4.1.1. Shield

It is like a small monitor placed after RL agent so the shield monitors the environment and the actions of the agent and whenever the agent wants to do something unsafe to the environment, the shield overwrites the unsafe action with a safe one. First, the system specifications are given as temporal logic. Then, the safety requirements are converted into an automaton that only visits the safe states F: $\varphi^s = (Q, q_0, \Sigma, \delta, F)$. Then the environment abstraction, which is modeled as MDP, is converted into an automaton $\varphi^M = (Q, q_0, \Sigma, \delta, F)$. Then finally the shield can be used to enforce safety specification φ^s by solving a safety game made up of safety specification φ^s and environmental abstraction φ^M, which can be won only if the system visits safe states F. By doing that, we have a correct-by-construction guarantee so that we know that this shield is correct[12]. So, if we attach it to the learning agent, the entire setting gets correct and then we can prove that our learning setting always adheres to the safety-critical specification.

4.1.2. Adaptation Engine

On top of the managed systems, there is a self-adaptive system, where we switch the shield at run time. An external subsystem corresponding to the adaptation engine is needed to be designed. Inside of this adaptation engine, it consists of Monitor, Analyze, Plan and Execute with Knowledge model[5] as you can see in Fig. 3.

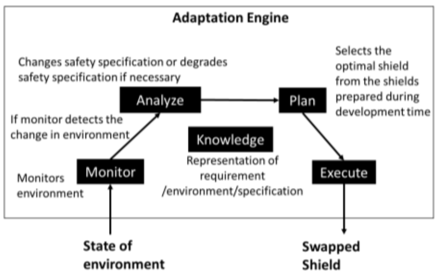
Adaptation Engine
Changes safety specification or degrades safety specification if necessary
Selects the optimal shield from the shields prepared during development time
Analyze
Plan
If monitor detects the change in environment
Knowledge
Representation of requirement /environment/specification
Monitors environment
Monitor
Execute
State of environment
Swapped Shield

Fig. 3: Adaptation Engine.

4.2. Shield synthesis, graceful degradation, and progressive enhancement

In the section of motivating example, we saw the water tank example and how the single shield performs. But for our scenario, we consider a situation that will happen if there is a malfunction in the water tank when the water level in the tank is just in level 5. We want to create a strategy (C4) that guarantees the goal property (G4). Such a strategy makes some assumptions as shown in Fig. 4. Then if the assumptions are violated, there is a slightly weaker goal (G3). The strategy (C3) for this goal makes weaker assumptions. If this also fails, there is a weaker goal (G2). The equivalent strategy (C2) for this goal makes weaker assumptions. If this also fails, there is even a weaker goal (G1). The corresponding strategy (C1) for this goal makes even weaker assumptions. For each goal, the shield will be synthesized and will prevent the water level to go above or below the specified level.

Assume the system is executing strategy C3 and one of its assumptions is violated, preventing G3 from being accomplished. A common method is to monitor the environment and notice when assumptions are breached. We would like the system to be built in such a way that it can continue from where it is currently with a strategy for the weaker G2, or, in the worst-case situation, keep G1 running until an engineer can solve the problem. In other words, when environmental assumptions are broken, the system's functioning should gracefully degrade[4]. We had like the system to try to provide G3 again after a period of degraded functionality, rather than needing to reset or stop the system completely because the assumptions of C3 are likely to hold most of the time in this instance. This is progressive enhancement[4].

4.3. During Development time

As shown in Fig. 4, we can see the goals from high-level goals to low-level goals. Goals which are like the safety specification, each goal or the safety specification is represented in the temporal logic[7,16] specification. As we know that the shield is synthesized from the safety specifications, for each of the goals, there is a shield synthesized from each of the safety specifications. We can also see that each shield is represented in different shades of the same color to show that how strong or risk-taking is the shield. This means that the topmost shield is the strictest shield and the bottom-most shield is the risk-taking shield. Then, we can see that in each goal, corresponding assumptions are made. This is how the shield is prepared during the development time.

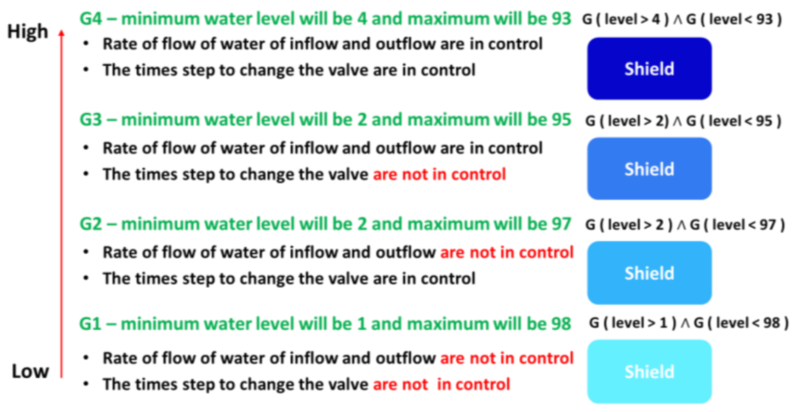
High
G4 – minimum water level will be 4 and maximum will be 93
G (level > 4) ∧ G (level < 93)
• Rate of flow of water of inflow and outflow are in control
• The times step to change the valve are in control
Shield
G3 – minimum water level will be 2 and maximum will be 95
G (level > 2) ∧ G (level < 95)
• Rate of flow of water of inflow and outflow are in control
• The times step to change the valve are not in control
Shield
G2 – minimum water level will be 2 and maximum will be 97
G (level > 2) ∧ G (level < 97)
• Rate of flow of water of inflow and outflow are not in control
• The times step to change the valve are in control
Shield
G1 – minimum water level will be 1 and maximum will be 98
G (level > 1) ∧ G (level < 98)
• Rate of flow of water of inflow and outflow are not in control
• The times step to change the valve are not in control
Shield
Low

Fig. 4: Hierarchical shield preparation during development time.

4.4. During Runtime

During runtime shown in Fig. 3, the monitor component monitors the environment and if it detects the change in the environment then, it will call the Analyze component. The Analyze component will change or degrade the safety specification for example from G (level > 4) ∧ G (level < 93) to G (level > 2) ∧ G (level < 95) if necessary. The Plan component selects the optimal shield from the list of shields that were prepared during development time. Finally, the Execute function swaps the shield. Now we will explain more on how the shield is swapped and how the Analyzer and Planner would swap the shields in the MAPE-k loop which is the adaptation engine in our hierarchical shielding architecture. So first, the analyzer checks whether the water level is greater than or equal to 4 and less than or equal to 93 and whether the inflow and outflow and the time step to change valve are in control. If this goal which is G4 is satisfied, it performs strategy C4. Else it goes to G3 and checks whether the water level is greater than or equal to 2 and less than or equal to 95 and whether the inflow and outflow is in control but not the time step to change the valve. If this goal is satisfied, it performs strategy C3. Else it goes to G2 and checks whether the water level is greater than or equal to 2 and less than or equal to 97 and whether the inflow and outflow is not in control but the time step to change valve is in control. If this goal is satisfied, it performs strategy C2. Else it goes to G1 and checks whether the water level is greater than or equal to 1 and less than or equal to 98 and whether the inflow and outflow and the time step to change valve is not in control. If this goal is satisfied, it performs strategy C1. Else it will fail. Then, the planner will select the optimal shields that were prepared during the development time.

But there is a question of how the safe and unsafe actions from MDP will be separated? It is done by the reactive system. The reactive system[2,13] separates the safe and unsafe action from MDP. A finite-state reactive system is a tuple $S = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$ with the input alphabet Σ_I, the output alphabet Σ_O, a finite set of states Q and the initial state $q_0 \in Q$. It is assumed that Σ_I is a product of Σ_I^1 and Σ_I^2 , that is $\Sigma_I = \Sigma_I^1 \times \Sigma_I^2$. Then, $\delta: Q \times \Sigma_I \to Q$ is a complete transition function, and $\lambda: Q \times \Sigma_I^1 \to \Sigma_O$ is a complete output function. Given the input trace $\overline{\sigma_I} = (\mathrm{x}_0^1, \mathrm{x}_0^2)(\mathrm{x}_1^1, \mathrm{x}_1^2)\ldots \in \Sigma_I^\infty$, the system S produces the output trace $\overline{\sigma_O} = S(\overline{\sigma_I}) = \lambda(q_0, \mathrm{x}_0^1)\lambda(q_1, \mathrm{x}_1^1)\ldots \in \Sigma_O^\infty$, where $q_{i+1} = \delta(q_i, (\mathrm{x}_i^1, \mathrm{x}_i^2))$ for all $i \geq 0$. The input and output traces can be merged to the trace of S over the alphabet $\Sigma_I \times \Sigma_O$, which is defined as $\bar{\sigma} = ((\mathrm{x}_0^1, \mathrm{x}_0^2), \lambda(q_0, \mathrm{x}_0^1))((\mathrm{x}_1^1, \mathrm{x}_1^2), \lambda(q_1, x^1))\ldots \in (\Sigma_I \times \Sigma_O)^\omega$.So what the reactive system does is, given an initial set of states, the output will return certain values, which will determine if the transition is allowed, safe, or unsafe. That is the finite state reactive system in the shielding paper. The shield is derived from the reactive system.

5. Evaluation

Before getting into the evaluation and experiment, we have some research questions that are to be shown. The research questions are as follows :- RQ 1. How effective is the hierarchical shielding in maintaining the water level than the existing method? RQ 2. How can progressive enhancement be achieved from the weakest goal G1 to a stronger goal? RQ 3. How efficient and robust is the hierarchical shielding approach compared to the existing method?

So to answer these research questions, we designed the experimental setting in a way that will show the difference between our proposed method and the existing method. To answer the first research question, in one graph we show the water level on one axis and the time on the other axis. So we can see how the water level is throughout the simulation. In the other graph, we show the percentage on one axis and the water level on the other to see how effective our proposed method is. For the second research question, we show a graph indicating different goals as mentioned in the previous section, and we show how the progressive enhancement can be achieved when there is some failure. For the last research question, we show how our proposed method performs and how the existing work performs by comparing the percentage of the water level being maintained and by showing the graceful degradation and progressive enhancement being performed in our proposed method but the existing method is not being able to perform any degradation of the goals.

We created simulations for 4 different scenarios, inflow failure scenario, outflow failure scenario, valve failure scenario, and multi failure scenario (valve and outflow failure). In each of the simulations, there are 6 graphs from (a) to (f). The graph (a) shows the valve position whether it is open or close. The x-axis represents the simulation duration and the y-axis that represents whether the valve position is closed or open. The graph (b) represents the water flow graph where the x-axis represents time and the y axis represents two parts where the positive part represents inflow(blue bars) and the negative part represents outflow red bars. Inflow is limited to 1 to 2 units. Outflow is limited to 0 or 1 unit. As the water flow affects the level of the system, the representation of the level is generated in the graph (c). The x-axis represents time, and the y axis shows water level. In this graph, we can see G4, G3, etc., which are the goals. The graph (d) is for the existing shielding method. In this graph we can see safe and fail, indicating that the only safe level for this shield is water level 4, and the water level should not go less than 4. The graph (e) can be seen on the top right of the simulation that shows the cumulative water level for our proposed method. The water level is represented on the x-axis, while the percentage is shown on the y axis in this graph. In the graph (f), we can see the cumulative water level for the existing shielding method.

5.1. Inflow Failure Scenario

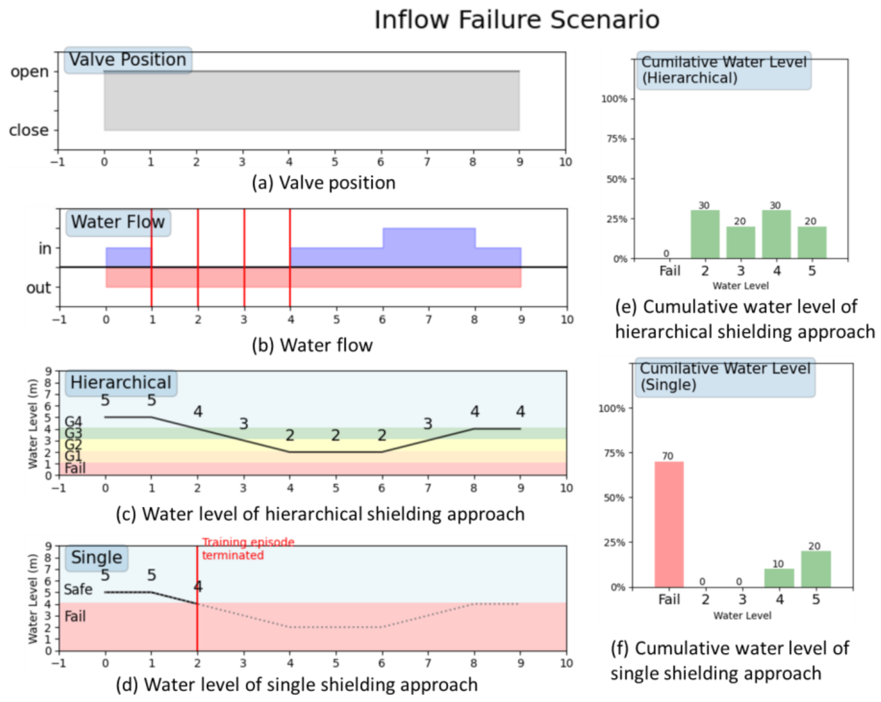

Fig. 5: Inflow Failure Scenario: (a) valve position, (b) waterflow, (c) water level of hierarchical shielding approach, (d) water level of single shielding approach, (e) cumulative water level of hierarchical shielding approach, (f) cumulative water level of single shielding approach.

In the inflow failure scenario shown in Fig. 5, the valve is open throughout the simulation. In the Fig. 5(b), we can see red lines indicating between the time 1 to 4 which means that even though the valve is open, there is no inflow happening, which is a violation of the criteria. This is the main thing to be noted in this scenario. We know that the maximum level of the water tank is 100 but we are considering the case when the water level is 5. The Fig. 5(c) shows how our method, hierarchical shielding works. In time steps 1, 2, 3, and 4, we could see in the Fig. 5(b) that due to the malfunction of inflow, there is not any water coming into the tank. Which means that the water level is decreasing. So in the Fig. 5(c), we can see that our proposal which is with the hierarchical shielding with MAPE-K, because of the relaxation of our goals, even though the water level is 3, the agent continues to learn and understands the system. We see over time, the water level drops to 2, and when the inflow and outflow change in such a manner, that our level starts increasing again, our system is able to gracefully recover from the scenario.

We can see that although the inflow failure is happening, it goes to different goal criteria, and it also gracefully degrades from G4 to G3 or G2 and also recovers or progressively enhances from G3 to G4 as per the status change. In the Fig. 5(d), the moment the water level of the system drops below 4, we see that the system stops, it fails. This is because the criteria mentioned for the shield safety mechanism indicate that anything below 4 is considered a failure and the agent has to restart the new episode. In this case, as the inflow is not working the water level goes below 4, so in this case, the moment it fails, it switches to fail. The moment the agent has failed, it will never come back to safe again. In the Fig. 5(e), we can see that among the ten-time points in the Fig. 5(c), 20% of the time, our proposed method is able to keep the water level at level 5, 30% of the time, at level 4, 20% of the time, at level 3 and 20% of the time, at level 2. From the Fig. 5(f), we can see that 20% of the time, the existing method is able to keep the water level at level 5, 10% of the time, at level 4 but unfortunately, the rest of the 70% of the time, fails and keeps the water tank to run dry.

5.2. Outflow Failure Scenario

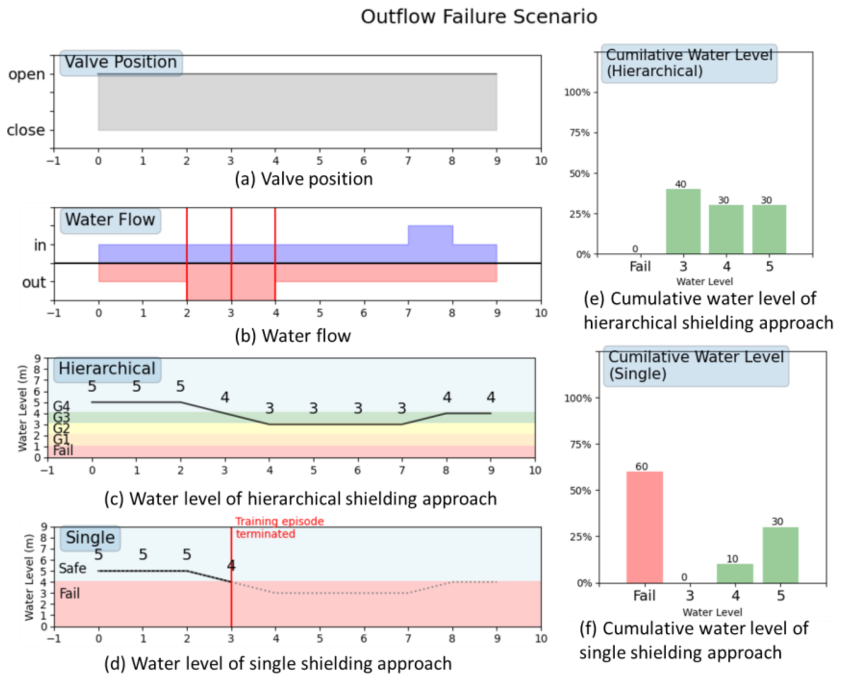

Fig. 6: Outflow Failure Scenario: (a) valve position, (b) waterflow, (c) water level of hierarchical shielding approach, (d) water level of single shielding approach, (e) cumulative water level of hierarchical shielding approach, (f) cumulative water level of single shielding approach.

In the outflow failure scenario shown in Fig. 6, the valve is open throughout the simulation and you can see the grey bar is open throughout the animation, throughout the time of the scenario. In Fig. 6(b), red lines are indicated in times 2, 3, and 4. If we see there, the outflow exceeded -1 which is the limit. This means that, due to the malfunction or the failure of the outflow, it is going more than how much it is permitted.

In time steps 2, 3, and 4, we could see in the Fig. 6(b) that due to the malfunction of outflow, it is faster than the inflow, which leads the water level to decrease. So in the Fig. 6(c), we can see that, in our proposed method, because of the relaxation of our goals, even though the water level decreases to 4 and 3, the agent continues to learn and understands the system.

When the outflow starts working properly again, it changes in such a manner, that our level starts increasing again, our system is able to gracefully recover from the scenario. We can see that although the inflow failure is happening, it goes to different goal criteria, and it also gracefully degrades from G4 to G3 and also recovers or progressively enhances from G3 to G4 as per the status change. Similarly, in Fig. 6(d), the moment the water level of the system drops below 4, we see that the system stops, it fails. Similar to the inflow failure scenario, in the Fig. 6(e) and Fig. 6(f), we can see that our proposed method is able to maintain the water level whereas, in the existing method, it fails and keeps the water tank running dry.

5.3. Valve Failure Scenario

In the valve failure scenario shown in Fig. 7, in the Fig. 7(a), we can see a red line, which is where the failure event has happened. So in our case, at time 2, the valve is triggered. The dotted lines represent, if the valve has not failed, that is when it should have opened. But it got delayed by a one-time unit due to the valve failure. Instead of the valve trying to start at time 3, it got delayed. In Fig. 7(b), we can see that from time 0 to 6 the inflow is 0 because the valve is only open at time 6 and outflow is -1. This means that there is only outflow and no inflow between the time 0 to 6.

In the Fig. 7(c), the water level is decreasing between the time 0 to 6 as there is no inflow due to valve failure. We can see that, in our proposal, because of the relaxation of our goals, even though the water level decreases to 4, 3, 2, and even 1, the agent continues to learn and understands the system. At time 7 when the inflow starts, it changes in such a manner, that our level starts increasing again, our system is able to gracefully recover from the scenario. We can see that although the inflow failure is happening, it goes to different goal criteria, and it also gracefully degrades from G4 to G3 to G2 to even G1 and also recovers or progressively enhances from G1 to G2 to G3 to G4 as per the status change. Similarly, in the Fig. 7(d), the moment the water level of the system drops below 4 at time 2 and after, we see that the system stops, it fails. In the Fig. 7(e) and Fig. 7(f), we can see that our proposed method is able to maintain the water level whereas, in the existing method, it fails and keeps the water tank to run dry.

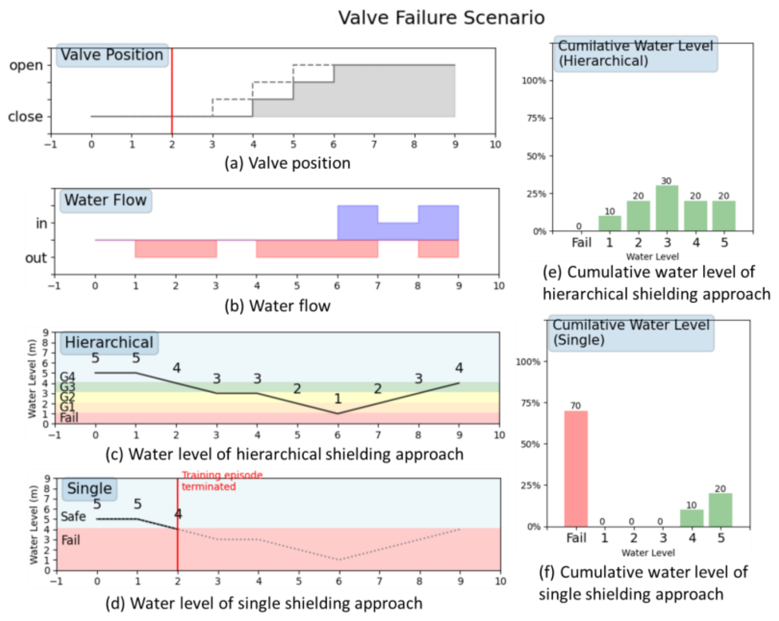

Fig. 7: Valve Failure Scenario: (a) valve position, (b) waterflow, (c) water level of hierarchical shielding approach, (d) water level of single shielding approach, (e) cumulative water level of hierarchical shielding approach, (f) cumulative water level of single shielding approach.

5.4. Multi Failure Scenario (Valve and Outflow Failure)

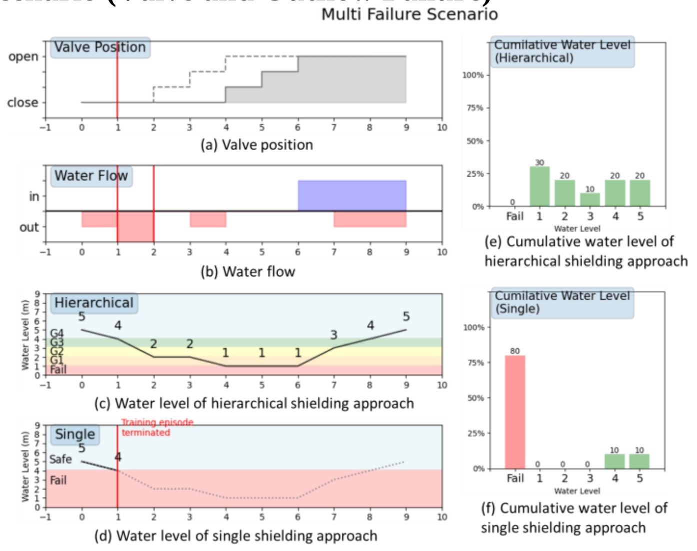

Fig. 8: Multi Failure Scenario: (a) valve position, (b) waterflow, (c) water level of hierarchical shielding approach, (d) water level of single shielding approach, (e) cumulative water level of hierarchical shielding approach, (f) cumulative water level of single shielding approach.

In the multi failure scenario shown in Fig. 8, we can see a red line in the Fig. 8(a), which is where the failure event has happened. So in our case, the valve should have triggered it at time 1 and the dotted line shows how it should have opened. But due to the valve fail, It tries to open at 4 and it opens the valve after 3-time steps which is at time 6. We also have outflow failure in this multi-failure scenario. we can see red lines indicating in time 1 and 2 in Fig. 8(b). We can see the outflow exceeded -1 which is the limit. This means that, due to the malfunction or the failure of the outflow, the outflow is going more than how much it is permitted.

In time steps 1 and 2, we could see that due to the malfunction of outflow, it is faster than the inflow. This means that the water level decreased from water level 4 to water level 2 due to no inflow during that time and the outflow being faster than usual. So in the Fig 8. (c), we can see that, in our proposal, because of the relaxation of our goals, even though the water level decreases to 4 to 2 to 1, the agent continues to learn and understands the system. When the outflow starts working properly again and when the valve opened at time 6, it changes in such a manner, that the water level starts increasing again, our system is able to gracefully recover from the scenario. We can see that although the inflow failure is happening, it goes to different goal criteria, and it also gracefully degrades from G4 to G2 to G1 and also recovers or progressively enhances from G1 to G3 to G4 as per the status change. Similarly in Fig. 8(d), the moment the water level of the system drops below 4, we see that the system stops, it fails at time 1. In the Fig. 8(e) and Fig. 8(f), we can see that our proposed method is able to maintain the water level whereas, in the existing method, it fails and keeps the water tank to run dry.

6. Discussion

6.1. RQ 1. How effective is the hierarchical shielding in maintaining the water level than the existing method ?

To answer this research question, we need to show how our proposed method, hierarchical shielding maintains the water level better than the existing method. In multi failure scenario shown in Fig. 8, we saw that in time steps 1 and 2, due to the malfunction of outflow, it is faster than the inflow in Fig. 8(b). This means that the water level decreased from water level 4 to water level 2 due to no inflow during that time and the outflow being faster than usual. So in Fig. 8(c), we saw that, in our proposal, because of the many levels of goals and the relaxation of our goals, even though the water level decreases from 4 to 2 to 1, the agent continues to learn and understands the system. When the high-level goal or assumption was about to be violated, it activated the next level goal and tried its best not to fail, unlike the existing method that just fails after it reaches the water level 4. Then when the outflow starts working properly again and when the valve opened at time 6, it changes in such a manner, that the water level starts increasing again, our system is able to gracefully recover from the scenario. So, until that our proposed method was able to maintain the water level instead of letting the water tank run dry. So, compared to the existing method, our proposed method is more effective in maintaining the water level.

6.2. RQ 2. How can progressive enhancement be achieved from the weakest goal G1 to a stronger goal ?

To answer this research question, we need to show how our proposed method progressively enhances from its weakest goal G1 to stronger goals G2 or G3, or G4. To show this, we consider the valve failure scenario shown in Fig. 7. In this scenario, we can see that there is a valve failure shown in Fig. 7(a). The valve should have opened at time 5, but due to the valve failure, it opens at time 6. So, there is no inflow happening until time 6 shown in Fig. 7(b). But we can see there is outflow until that time. But the problem is the water level is just at level 5. But because there is outflow and no inflow until time 6, there is a decrease in the water level. In Fig. 8(c), we can see that until time 6, due to the decrease in the water level, the agent is at the weakest goal G1. When the inflow has started working from time 6, the water level is maintained. Then, after a period of degraded functionality, the system is would like to start providing the stronger goals again. So, from time 6 to 9, we can see that the system has started to provide stronger goals. So basically, this is how a progressive enhancement is achieved from the weakest goal.

6.3. RQ 3. How efficient and robust is the hierarchical shielding approach compared to the existing method ?

To answer this research question, we will consider all the scenarios. To demonstrate how efficient and robust the system is using our proposed method, we saw how the hierarchical shield can maintain the water level and how our method performs graceful degradation and progressive enhancement, whereas the existing method was not able to do any graceful degradation in any of the four cases. That is because, the existing method, which was the single shielding method, it only has one goal and when there is a change in the environment, it cannot change its goal. So, that is the reason why it fails the moment when the goal is violated. But the hierarchical shielding method allows you to investigate conditions that were previously deemed dangerous. However, it was not inherently dangerous, and unlike the previous shielding method, our idea does not squander it. It's more efficient and it learns more by entering slightly more dangerous, higher-risk states, while still gaining the benefit of gaining more knowledge.

The disadvantage of our proposal is that extra effort from engineers is required to develop additional requirements for environmental change. When the safe state is on the verge of failing that our approach falls short. It is difficult to apply our solution in the previous shielding if the water level is only 1, as taking more risks than 1 is pointless. Where the difference between a safe failure and an actual failure is small, our proposal will fail. However, if there is a sufficient gap between safe failure and actual failure, our proposed method works fine.

7. Related Work

The work [2] proposes a novel method for learning optimum rules while enforcing temporal logic features. They propose to synthesize a reactive system termed a shield based on the temporal logic specification that the learning system must follow. Depending on where the shield is used, the shield is integrated into the regular reinforcement learning process in two different ways. The shield acts in the first one whenever the learning agent is about to make a decision and gives a selection of safe options. The shield is also introduced after the learning agent in a second approach. The learner's actions are monitored by the shield, which corrects them only if the chosen action violates the specification. So, in essence, they take a formal techniques approach to the problem of ensuring reinforcement safety. In addition, the study [3] proposes a similar approach for reinforcement learning safety. They offer a strategy in this research that uses runtime monitoring to avoid reinforcement learning agents from doing incorrect behaviors and leveraging existing knowledge to intelligently explore the environment. Each monitor has a property and a context that they want to enforce on the agent. A meta-monitor adapts the monitors by dynamically activating and deactivating them depending on the situation in which the agent is learning.

The majority of adaptive system techniques rely on models, behavior, or architecture models to characterize the system and its environment. One of the challenges in developing such models is the uncertainty surrounding their correctness and completeness. As a result, the engineers make assumptions that may turn out to be incorrect later on. As a result, the work in [3] establishes a systematic, layered framework for merging behavior models, each with its own set of assumptions and dangers. Through approaches like discrete controller synthesis[8,9], these models are used to build operational strategies, which are subsequently implemented concurrently at runtime. They demonstrate how their framework may be used to modify the system's functional behavior via graceful degradation when a higher-level model's assumptions are broken, and progressive enhancement when those assumptions are satisfied or restored.

Smart systems, for example, are built on constant interaction with the dynamic and largely unknown environment in which they are deployed. Traditional development methods based on varying environmental circumstances are no longer viable. Modern methodologies, on the other hand, should be able to create systems that can learn how to act in a variety of environments on their own. Machine learning approaches enable systems to learn how to carry out a series of activities to attain a specific goal. When anything changes, the system can learn new policies and techniques for executing activities on its own. This flexibility has a price: the developer no longer has complete control over the system's behavior. As a result, there's no way to guarantee that the system won't break crucial properties like safety-critical ones. To address this problem, [10] proposed that machine learning approaches be integrated with appropriate reasoning mechanisms aimed at ensuring that the machine learning algorithm's decisions do not violate safety-critical constraints. They offer a method for detecting violations of system invariants in action execution policies that combines machine learning with runtime monitoring. So, in general, smart systems must be capable of continuously perceiving the environment in which they function, identifying changes, and responding to those changes. As a result, they believed that modern smart system development methodologies should rely on strategies that allow systems to learn how to act in a variety of environments on their own. Machine learning approaches can also learn how to act on a running system to achieve a desired goal on their own. Rather than logic programs with predetermined rules, these strategies rely on models educated on data and examples. The programmer is replaced by a computer that can update its models when fresh data from the environment becomes available. Machine learning models, once trained, enable for effective change handling, which means that when a change occurs, the system learns new policies for action execution on its own. The use of machine learning dramatically alters how software systems are produced, implying that the choice of which actions to perform in various environmental conditions is no longer in the hands of the developer, but rather is made automatically.

They utilize reinforcement learning in this study because it is a powerful machine learning technique for making decisions. Machine learning approaches give automatic support, transferring control from developers to the system. As a result, the developer can no longer guarantee that the system will not break crucial features, such as invariants, which can be used to describe safety-critical properties, for example. As a result, they argue that while systems must be self-adaptive, intelligence must also be kept under control. They propose a novel method, termed WiseML, for ensuring that machine learning decisions do not violate a set of important attributes. This method integrates machine learning, notably reinforcement learning[11], with run-time monitoring approaches to ensure that essential safety-critical requirements are maintained. On the one hand, WiseML uses machine learning techniques to allow systems to adapt. WiseML, on the other hand, uses run-time monitoring to ensure that the rules recommended by reinforcement learning do not break a set of safety-critical conditions.

We adopt all these ideas and they are all close to my research context like the reinforcement learning algorithms or MDP based reinforcement learning algorithms and also safe shielding approach for reinforcement learning algorithms. Also they were similar from the viewpoint of ensuring safety requirements in the reinforcement learning based approach.

8. Conclusion

Our research introduces an approach for safe reinforcement learning by combining the hierarchical shielding approach with self-adaptive techniques which can be used for AI applications or machines in the industries that are using reinforcement learning. Hierarchical shielding has different levels of safety requirements, that have different levels of shield that will be chosen and swapped during the run time. For choosing and swapping the appropriate shield, we use the self-adaptive techniques to achieve hierarchical shielding by using graceful degradation and progressive enhancement.

The finite state reactive system has only one set of specifications with a fixed goal that tells which is the safe and unsafe state. But for many goals, defining many reactive systems and choosing one is not so trivial. So we need something more than a reactive system that is more complex. We require something akin to a multi-goal reactive system, which will be explored more in the future.

9. Acknowledgements

The research was partially supported by JSPS KAKENHI and JST Research Fellowship W-SPRING.

10.References

[1] R. Chopra and S. S. Roy, “End-to-end reinforcement learning for self-driving car,” *Advances in Intelligent Systems and Computing*, pp. 53–61, 2020.

[2] M. Alshiekh and R. Bloem and R. Ehlers and B. K{\"o}nighofer and S. Niekum and U. Topcu, “Safe Reinforcement Learning via Shielding,” AAAI 2018, pp. 2669-2678, 2018.

[3] P. Mallozzi, E. Castellano, P. Pelliccione, G. Schneider, and K. Tei, “A runtime monitoring framework to enforce invariants on reinforcement learning agents exploring complex environments,” *2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE)*, pp. 5–12, 2019.

[4] A. Back and R. Arnold and T. Quill, “Hope for the Best, and Prepare for the Worst, ” Annals of Internal Medicine, 2003, 138, pp. 439-443, 2003.

[5] P. Arcaini, E. Riccobene, and P. Scandurra, “Modeling and analyzing MAPE-K feedback loops for self-adaptation,” *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 13–23, 2015.

[6] Saunders, William et al. “Trial without Error: Towards Safe Reinforcement Learning via Human Intervention.” AAMAS (2018), pp. 2067-2069, 2018.

[7] A. Pnueli, “The temporal logic of programs,” 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), pp. 46–57, 1977.

[8] L. Nahabedian, V. Braberman, N. Dippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel, “Dynamic update of discrete event controllers,” *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1220–1240, 2020.

[9] L. Nahabedian, V. Braberman, N. D'Ippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel, “Assured and correct dynamic update of controllers,” *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 96–107, 2016.

[10] P. Mallozzi, P. Pelliccione, and C. Menghi, “Keeping intelligence under control,” *Proceedings of the 1st International Workshop on Software Engineering for Cognitive Services*, pp. 37–40, 2018.

[11] P. Verma and S. Diamantidis, “What is reinforcement learning? – overview of how it works,” *Synopsys*, 27-Apr-2021. [Online]. Available: https://www.synopsys.com/ai/what-is-reinforcement-learning.html.

[12] M. Wen, R. Ehlers, and U. Topcu, “Correct-by-synthesis reinforcement learning with temporal logic constraints,” *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4983–4990, 2015.

[13] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang, “Shield synthesis:” *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 533–548, 2015.

[14] García, Javier and F. Fernández. “A comprehensive survey on safe reinforcement learning.” J. Mach. Learn. Res. 16 (2015), pp. 1437-1480, 2015.

[15] D. Sinreich, “For autonomic computing. an architectural blueprint,” 2006. [Online]. Available: https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf. [Accessed: 17-Jan-2022].

[16] L. Goble and A. ter Meulen, “Logic and Natural Language,” in *The Blackwell Guide to Philosophical Logic*, 1st ed., Malden: Blackwell, 2001, pp. 461–483.

[17] N. Bäuerle and U. Rieder, “Markov Decision Processes,” *Jahresbericht der Deutschen Mathematiker-Vereinigung*, vol. 112, pp. 217–243, 2010.