Thesis for The Degree of Licentiate of Engineering

Engineering Trustworthy Self-Adaptive Autonomous Systems

Piergiuseppe Mallozzi

Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2018

Engineering Trustworthy Self-Adaptive Autonomous Systems

Piergiuseppe Mallozzi

Technical Report No 186L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2018.

Abstract

Autonomous Systems (AS) are becoming ubiquitous in our society. Some examples are autonomous vehicles, unmanned aerial vehicles (UAV), autonomous trading systems, self-managing Telecom networks and smart factories. Autonomous Systems are based on a continuous interaction with the environment in which they are deployed, and more often than not this environment can be dynamic and partially unknown. AS must be able to take decisions autonomously at run-time also in presence of *uncertainty*. Software is the main enabler of AS and it allows the AS to *self-adapt* in response to changes in the environment and to *evolve*, via the deployment of new features.

Traditionally, software development techniques are based on a complete description at design time of how the system must behave in different environmental conditions. This is no longer effective since the system has to be able to *explore* and *learn* from the environment in which it is operating also after its deployment. Reinforcement learning (RL) algorithms discover policies that can lead AS to achieve their goals in a dynamic and unknown environment. The developer does not specify anymore how the system should act in each possible situation but rather the RL algorithm can achieve an optimal behaviour by *trial and error*. Once trained, the AS will be capable of taking decisions and performing actions autonomously while still learning from the environment. These systems are becoming increasingly powerful, yet this flexibility comes at a cost: the learned policy does not necessarily guarantee safety or the achievement of the goals.

This thesis explores the problem of building trustworthy autonomous systems from different angles. Firstly, we have identified the state of the art and challenges of building autonomous systems, with a particular focus on autonomous vehicles. Then, we have analysed how current approaches of formal verification can provide assurances in a System of Systems scenario. Finally, we have proposed methods that combine formal verification with reinforcement learning agents to address two major challenges: how to *trust* that an autonomous system will be able to achieve its goals and how to *ensure* that the behaviour of AS is safe.

Keywords

Autonomous Systems, Automotive, System Trustworthiness, Formal Verification, Machine Learning, Runtime verification, Monitoring and enforcement.

Acknowledgment

My biggest thank goes to Patrizio Pelliccione for being the perfect supervisor and a life mentor. Thank you for always encouraging me and for pushing me on the right path. Your positive attitude inspires me to focus on what is important and helps me be more confident. Thank you for always supporting my choices and for giving me the freedom to explore different research topics.

To my co-supervisor Christian Berger for his good advice and support. To my examiner Ivica Crnkovic for always being there for me and for being a great leader.

I want to thank my colleagues at the Computer Science and Engineering department for welcoming me in such a great working environment. It doesn't matter in which side of the river I am, fika-time is always a great moment to exchange ideas, chat with friends and even start new research collaborations. In particular, I would like to thank my co-authors Raúl Pardo and Gerardo Schneider, I look forward to continuing our collaboration. A special acknowledgement also to my office mates Katja, Sergio and Rebekka for always making the office a fun place to work.

To the Cougar Boys, Traveling Lingonberries, Pistachio Blues, The Band or whatever our name is now, for keep rocking during the rainy days in Gothenburg. Thank you Grischa, Marco, Carlo, Enzo, and Evgeni for bearing with me playing the same two chords for all the 50 bars (I counted them) of Riders on the Storm solo.

This work would not have been possible without the support of the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. I can not believe how lucky I am for being part of such an ambitious program. I will always be grateful for all the opportunities that you have been offering me and I look forward to starting my exchange at Berkeley soon!

To the Japan Society for the Promotion of Science (JSPS) for giving me one of the best summers of my life, this experience has really *changed me.* Having the possibility to visit Japan and work with other research groups gave a whole new perspective to my work and boosted my research in directions that I did not see before.

Finally, I would like to express my deepest gratitude to my family and all my friends for making me the person who I am today. You kept my life more or less balanced during all these years.

List of Publications

Appended publications

This thesis is based on the following publications:

A P. Mallozzi, P. Pelliccione, A. Knauss, C. Berger, and N. Mohammadiha "Autonomous vehicles: state of art, future trends, and challenges" book chapter in *Automotive Software Engineering: State of the Art and Future Trends*, 2017, Springer

B P. Mallozzi, R. Pardo, V. Duplessis, P. Pelliccione, and G. Schneider "MoVEMo - A structured approach for engineering reward functions" in *International Conference on Robotic Computing (IRC)*, Laguna Hills (CA), 2018, IEEE

C P. Mallozzi, M. Sciancalepore, and P. Pelliccione "Formal verification of the on-the-fly vehicle platooning protocol" in *Software Engineering for Resilient Systems (SERENE), Gothenburg (Sweden)*, 2016, Springer

D P. Mallozzi, P. Pelliccione, and C. Menghi "Keeping intelligence under control" in *International Workshop on Software Engineering for Cognitive Services (SE4COG), Gothenburg (Sweden)*, 2018, ACM

E P. Mallozzi, E. Castellano, P. Pelliccione and G. Schneider "Using run-time monitoring to address safe exploration for reinforcement learning agents" *in submission*

Other publications

The following publications were published during my PhD studies. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

F P. Pelliccione, E. Knauss, R. Heldal, S. M. gren, P. Mallozzi, A. Alminger, and D. Borgentun "Automotive architecture framework: The experience of Volvo cars"
in *Journal of Systems Architecture (JSA)*, 2017, Elsevier

G P. Pelliccione, E. Knauss, R. Heldal, M. Ågren, P. Mallozzi, A. Alminger, and D. Borgentun "A proposal for an automotive architecture framework for Volvo Cars"
in *Automotive Systems/Software Architectures (WASA)*, Venice (Italy), 2016, IEEE

H P. Mallozzi "Combining machine-learning with invariants assurance techniques for autonomous systems"
in *International Conference on Software Engineering Companion (Doctoral Symposium)*, 2017, IEEE

Works in progress

The following works are to be submitted for publication.

I D. Brugali, P. Mallozzi, G. Badaro "Autonomous Vehicles - Managing Variability"

J P. Mallozzi, E. Castellano, P. Pelliccione, K. Tei "Combining controller synthesis with reinforcement learning"

K P. Mallozzi, E. Castellano, P. Pelliccione, G. Schneider "Self-adaptive safety monitors for reinforcement learning agents"

Research Contribution

I took the lead in all the included papers.

My contribution in Paper A is the state of the art and challenges to guarantee safety in autonomous vehicles and the future directions. The remaining authors have contributed with reviews and short paragraphs.

In Paper B I have designed the method, set up the experiment and collected and analysed the data. The implementation was partially realised by me and partially by the third author. The remaining authors contributed to reviewing the paper, writing paragraphs related to the run-time monitoring and external tools support.

The implementation, evaluation of the protocol and the formalisation of the properties described in Paper C was done entirely by me. The remaining authors supported me with the formal verification of the properties in randomly generated scenarios and by reviewing the paper.

Paper D builds up on my ideas, the remaining authors helped me with the definition of the challenges and the polishing of the paper.

Paper E is the realization of the overall idea described in Paper D. Most of the contribution of the paper is the result of my work. I have designed the method and implemented the approach. The remaining authors have helped me launching the experiments and collecting data.

Contents

Chapter 1

Introduction

1.1 Introduction

Autonomous Systems (AS) are becoming ubiquitous in our society. Autonomous vehicles, unmanned aerial vehicles (UAV), autonomous trading systems, self-managing telecom networks, smart factories can be all considered Autonomous Systems. In the near future, we will assist in systems exhibiting higher and higher levels of autonomy that will put new demands on the engineering of such systems.

Some communities refer to autonomous systems as *self-adaptive systems*, i.e. systems that are able to adapt their behaviour at runtime without human intervention [1–3] in response to changes in the environment or in their internal state. They implement some sort of feedback loop to perform their adaptations [4]. A self-adaptive system gathers observations from the environment, processes them considering also its internal state and adapts itself to achieve its goals.

Classical software development techniques require to fully describe the system behaviour in each possible environmental condition. The developer models different configurations at design-time that are then activated at runtime according to the events coming from the environment [5]. This is becoming unpractical — if not even impossible — in AS where there is a high, or even infinite, number of environmental conditions to be considered for adaptation. For this reason, modern development techniques for AS must rely on techniques that allow creating systems that autonomously *learn* how to behave in different environmental conditions.

AS are intrinsically based on a strong interaction with their environment. Such an environment is usually unknown, meaning that AS have to operate under high *uncertainty* [6–8]. Sources of uncertainty could be *external* to the system, such as the environment in which the software is deployed, the availability of the resources that the system can access at a given time or the difficulty of predicting the other systems behaviour. Other sources of uncertainty can be *internal* to the system, such as the inability of the system to predict the impact of its adaptations on the environment. Uncertainty can require the system to dynamically change its goals and adapt itself while it is running.

Reinforcement learning (RL) [9] has been successfully used as a technique to act in an unknown environment in order to achieve a goal. It enables the software agent to autonomously learn to perform actions by *trial and error.* As with self-adaptive systems, it is based on a feedback loop where the software agent performs actions on the environment in response to the observations, receiving a numerical value (*reward*) for each action. The goal of the RL agent is to maximise the cumulative received reward. After training, the RL agent can effectively handle changes i.e., when a change occurs the system autonomously learns new policies for actions execution.

The use of reinforcement learning as the main driver of the system's adaptation introduces flexibility in terms of *explore* and *acting* in an unknown environment but it also poses new challenges. The choice of which adaptation to perform in the environment is no longer in the developer's hands but is rather automatic. When dealing with safety-critical systems one major challenge is the assurance *safety.*

The use of formal methods is often seen as a way to increase confidence in a software system. Mathematical techniques can be used to reduce errors in a software system. Methods such as *model checking* and *theorem proving* can formally proof the compliance of a model with some formally specified properties. When dealing with self-adaptive systems applying traditional techniques at design time is not enough to certify the compliance of the deployed system with its requirements. This is due to the fact that in AS the requirements are dynamic and subject to change at runtime [10], so at least part of the analysis must be shifted at runtime as well. Techniques such as *runtime verification* (RV) [11, 12] can be used to monitor software executions. It can then detect violations of safety properties at runtime and it provides the possibility of reacting to the incorrect behaviour of the software agent whenever an error is detected.

Our research goes in the direction of building ***trustworthy*** **self-adaptive software systems** with particular emphasis on *reinforcement learning* to drive the system adaptations. In the context of this thesis *trustworthiness* means assuring the correct inference of the goal to the autonomous software agent and assuring safety during its execution in scenarios affected by uncertainty.

In conclusion, this thesis: (i) analyses the state of the art, the challenges, and future trends in **Autonomous Systems**, with autonomous vehicles as the main area, and (ii) proposes methods for addressing *trustworthiness* that combine **Formal verification** (*model-checking* and *runtime monitoring*) techniques to formally assure the compliance of the system with safety-critical properties with **Reinforcement learning**, used as the main enabler for the decision-making process of the autonomous system.

1.2 Background and Related Works

1.2.1 Self-Adaptive Systems

Traditionally software is developed by programmers who craft it according to the decisions they made. Modern systems are increasingly requested to operate in dynamic and often unknown environments that are dominated by uncertainty. Consequently, it is extremely difficult to anticipate all the possible and subtle variations of the environment at design-time. Furthermore, in an environment with a lot of variability and uncertainty, it is often impossible to completely define safety cases and to follow traditional certification processes. In this context, systems are required to continuously adapt to changes that occur in the environment.

Self-adaptive systems implement some sort of feedback loop that drives their adaptations [4]. This basic mechanism for adaptation has been applied for years in control engineering; it consists of four main activities: collect, analyse, decide, and act. A feedback loop particularly important for decision-making is the OODA loop by Boyd [13] where we do not only predict what our system has to do but also what the external systems are going to be doing. Another well-known reference model for describing the adaptation processes is the MAPE-K loop (consisting of the parts Model, Analyse, Plan, Execute, and the Knowledge Base) [14]. Furthermore, often systems collect data from the environment, learn from them, and, consequently, continuously improve. An example of such a system is the Never-Ending Language Learning [15].

1.2.2 Reinforcement Learning

Reinforcement Learning (RL) [16] is a machine learning approach where a software agent can learn to *self-adapt* by exploring the environment. RL techniques search for optimal policies by performing actions and observing the effect of these actions in the underlying environment.

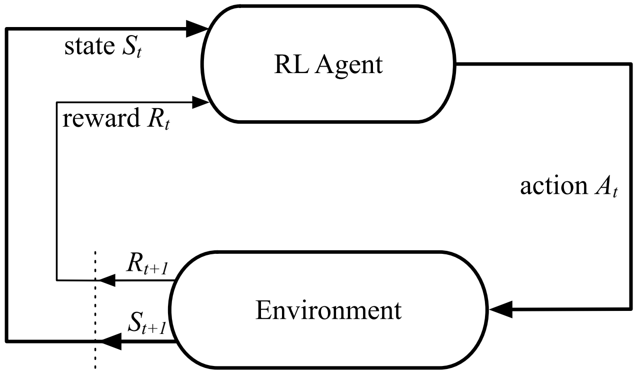

Figure 1.1: Reinforcement learning framework showing the interaction between the agent and the environment.

The agent and the environment interact in discrete time steps as shown in the Figure 1.1. At each time step s_t the agent receives observations from the environment that correspond to a certain *state* S_t. It then chooses an action A_t among the set of possible actions $\mathcal{A}$. The action is then applied to the environment that moves to a state S_{t+1} and returns a reward R_{t+1} to the agent. The environment where an RL agent acts can be formally described as a Markov Decision Process (MDP), which can be defined as follows.

Definition - Markov Decision Process (MDP).
An MDP is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma >$ where:

— $\mathcal{S}$ is a finite set of states

— $\mathcal{A}$ is a finite set of actions

— $\mathcal{T}$ is a state transition probability matrix $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$ that assigns the probability of the next state to be s' by taking action a from the current state s.

— $\mathcal{R}$ is the reward function. It assigns an immediate reward r_{t+1} after transitioning from state s to state s' due to action a. It is formalised as the expected reward at next time step: $\mathcal{R}^a_s = \mathbb{E}[r_{t+1} | S_t = s, A_t = a]$

— γ is the discount factor $\gamma \in [0, 1]$. It represents the difference in importance between future rewards and present rewards. At each step all future rewards are discounted by γ as follows: $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$

The MDP satisfies the *Markov Property*: the next state only depends on the current state and the chosen action, so it is conditionally independent of all previous states and actions. In other words: *“ the future is independent of the past given the present ”*

Definition - Return.
The *return* $G = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$ is the sum of all the *discounted* rewards that the agent will get from time step 0 infinitely into the future. The discount factor indicates the preference between receiving a short-term reward ($\gamma = 0$) versus a long-term reward ($\gamma = 1$).

Definition - Policy.
A policy π fully defines the behaviour of a RL agent. It can be either deterministic or stochastic. A stochastic policy is a distribution over actions given a state: $\pi(a|S) = \mathbb{P}[A_t = a | S_t = s]$. Otherwise, if the policy is deterministic it becomes a mapping function from state to action $\Pi : \mathcal{S} \to \mathcal{A}$.

The goal of the RL agent is to learn a policy that maximises the return. The *optimal policy* π^* is the one that, among all possible policies, maximises the expectation of the return: $\pi^* = max_{\pi \in \Pi} \mathbb{E}_\pi(G)$

The reward function.
A well-defined reward function has demonstrated to be successful in several cases such as Atari games [17] and board games [17]. These examples show

that a simple reward function, such as the score of the game, can teach the agent to achieve the optimal policy. In other methods such as apprenticeship learning, the reward function is learned from observations [18]. The idea is that we take as given an expert optimal policy and we determine the underlying reward structure.

Sometimes the agent needs an early feedback on the success of their actions without having to wait for the end of the task. Reward shaping has been addressing this topic by providing guidance to the agent and incorporating prior knowledge in the reinforcement learning [19, 20].

Learning.
Q-learning [21] is a simple and effective way to learn the optimal policy of an MDP. It estimates the long-term expected return of each action that can be executed from a given state. These estimations represent the *quality* of the actions at a particular state and are known as *Q-values*. These values are learned at each iteration by integrating the *reward* received from the environment in the update of the current Q-value estimate:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Where $\alpha \in [0, 1]$ is the *learning rate* and determines how much the newly acquired information overrides the previous knowledge of the Q-value and $\max_a Q(s_{t+1})$ is the the max Q-value over all the possible actions in the next resulting state s_{t+1}.

Deep Learning.
When the state space becomes too big to store a Q-value for each individual state-action pair an approximate model is needed. Neural networks can be used to approximate the Q-value and policy functions. Deep learning methods use neural networks composed of many layers between the input and output layers. Hence, instead of updating individual Q-values, now the updates are made to the parameters of the neural networks with techniques such has *stochastic gradient descent.*

Actor-critic algorithms such as the Deep Policy Gradient (DPG) algorithm [22], Deterministic Deep Policy Gradient (DDPG) [23] and A2C [24] use two neural networks: the actor and the critic. The actor is the policy $\pi_\theta(a|s)$ where θ are the parameters of the network. The critic instead estimates a value function $Q^w(s, a)$ and has parameters w. The actor conducts actions in the environment while the critic assists the actor in the learning. For example, in DDPG the critic estimates the value of the current policy by Q-learning and the actor updates its policy in a direction that improves the action-value function Q.

Partial Observability.
In real-world applications, an RL agent cannot be provided with the full state of the environment because this can also be impossible to determine. Usually, the agent perceives partial observations of the environment through its sensors. In the case of partial observations, the Markov assumption of the MDP is

not valid anymore. The true state of the environment is revealed gradually from several observations over time. Partially Observable Markov Decision Processes (POMDP) better capture the dynamics of real-world environments by assuming that what the agent receives are only some measurements (e.g. from its sensors) of the underlying system state.

Formally, a POMDP is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma >$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$ and γ are the states, actions, transition probabilities, reward function, and discount factor, respectively. However, this time the agent does not receive the true state of the system but instead it gets an observation $o \in \Omega$. Finally, $\mathcal{O}$ is an observation model that gives us the probability of seeing observation o in a state s: $\mathcal{O}(o|s)$.

In order to decode the underlying state of the POMDP, multiple observations have to be integrated over time so to gradually reveal the state of the system. Traditional neural networks cannot use information from previous states to infer next ones. Recurrent Neural Networks (RNN) address this issue by having loops they have the capacity to learn temporal dependencies. Long Short-Term Memory (LSTM) [25] are a particular kind of RNN that work very well in practice and have been successfully used in combination with deep learning algorithms such as Deep Q-Learning [26] and DPG [27] to solve a variety of tasks in partially observable environments.

In all our works we do not modify existing algorithms and techniques, but rather we use state-of-the-art approaches to validate general methods that can be applied to a variety of algorithms. In **Paper B**, we have worked on engineering the reward function and validated our results on an agent that uses DDPG as the learning algorithm. In **Paper E**, an agent has to move safely into a partially observable environment. We use the A2C algorithm in combination with LTSM in order to tackle the partial observations conveyed to the agent. In both papers, we use *reward-shaping* to modify the reward of the agent at runtime and guiding towards the goal.

1.2.3 Trustworthiness

Especially when dealing with safety-critical applications, it is essential to guarantee that AS act *safely* in the environment where they are deployed. In other words, the software must work in a reliable manner and must be safe for humans as life may depend on it. In this thesis, we refer to building *trustworthy self-adaptive systems* meaning to provide evidence that important aspects of the AS are correct. In presence of self-adaptation systems, the fulfilment of the requirements cannot be guaranteed completely at design-time but at least part of the assurance needs to be performed at runtime [10]. In this thesis we to provide *assurance* to the software system [28] by combining design-time modelling and verification, such as *model checking*, and runtime assurances techniques such as *runtime verification*.

Model Checking.
The main purpose of a model-checker is to verify a given model of a system with respect to a requirement specification by exhaustively and automatically checking all its states. The state space is a directed graph whose nodes encode

the states of the whole system and whose edges represent the state change. It ultimately represents all the behaviours of the modelled system by branching all possible ways the components can interact with each other. Because of the exploration of all states in a brute-force manner, model checking suffers of the *scalability* problem.

One possible way to represent the system to be verified is with timed-automata. Like the model, the requirement specification (or properties) to be checked must be expressed in a formally well-defined and machine-readable language. UPPAAL is a tool where one can model the system with timed-automata and the properties with a subset of TCTL (timed computation tree logic) [29, 30]. It allows the verification of safety and behavioural properties of a model, such as the absence of deadlocks or the propagation of a safety-critical message within a certain time. For example, the path formulae $A <> \varphi$ (or equivalently $\neg E[\,]\neg\varphi$) expresses that φ will be eventually satisfied, or more precisely, that in each path will exist a state that satisfies φ. The path formulae $A[\,]\varphi$ expresses that φ should be true in all reachable states.

Runtime Verification.
Runtime Verification (RV) [11, 12] is a lightweight verification technique based on monitoring software executions. It has its origins in *model checking* but it mitigates the state explosion problem by having a more scalable approach to software verification. It detects violations of properties, occurring while the monitored program is running, eventually providing the possibility of reacting to the incorrect behaviour of the program whenever an error is detected.

Properties verified with RV are specified using any of the following approaches: (i) annotating the source code of the program under scrutiny with *assertions* [31]; (ii) using a high-level specification language [32–34]; or (iii) using an automaton-based specification language [35–37].

One way to verify properties at runtime is through the use of *monitors*. A monitor is a piece of software that runs in parallel to the program under scrutiny, controlling that the execution of the latter does not violate any of the properties. In addition, monitors usually create a log file where they add entries reflecting the verdict obtained when a property is verified. In general, monitors are automatically generated from the annotated/specified properties [38].

On one hand, static verification approaches such as model checking can verify that the system is compliant with some important properties, but some systems are too complex to be model checked in practice. On the other hand, runtime verification can prove the satisfaction of the properties on large systems but only on the fraction that is executing during the verification. We use model checking in **Paper C** to verify that the protocol for on-the-fly vehicle platooning is compliant with the properties. We randomly generate different scenarios and check the properties on all of them. In **Paper B** we use UPPAAL to model and statically verify the reward function at design-time and runtime verification approaches to enforce it during the learning of the agent. Finally, in **Paper E** we model runtime monitors in the form of *patterns* [39] to verify properties on the RL agent itself.

1.3 Goals and Methodology

The overall goals of this thesis are:

G1: To investigate the state-of-the-art and current challenges in autonomous systems with the focus on the automotive domain.

G2: To propose concrete solutions on how to build trustworthy AS that can perform safely also in presence of uncertainty.

The goal **G1** aims to create a big picture of the problem with respect to Figure 1.2. It aims to investigate AS as entities that on one level - short loop - locally self-adapts at runtime. On another level - long loop - field data collected by a set of AS are exploited by developers to produce new versions of the software to be then deployed on the agents. **G2** aims to address issues *within* the autonomous systems, represented in the lower part of Figure 1.2. The autonomous systems act in an environment that is at best partially known, hence trusting the system that it will always act as intended is one of the challenges that need to be addressed.

For the first goal we have formulated the following research question:

RQ1: What is the state-of-the-art, the challenges and future trends in building Autonomous Systems in the automotive domain?

Research question **RQ1** has been addressed in Paper A with respect to autonomous vehicles. We have conducted a study of what is the state-of-the-art of autonomous vehicles with respect to the vehicle functionalists and the several vehicle software architectures. Current trends are in using machine learning to find solutions to several autonomous vehicles problems, from perception to decision-making. We have identified five challenges for guaranteeing safety in autonomous vehicles with the use of machine learning being a major one.

Methodology: We have conducted a literature review to understand the state-of-the-art of autonomous systems and the state-of-practice of the major automotive companies. We have sampled papers from a number of conferences and journals articles.

Due to the complexity of goal **G2**, we have broken it down into several research questions that build up in the direction of engineering trustworthy autonomous systems. One of the main problems is that it is impossible to predefine all the system's adaptations at design-time. This is because the environment where the system is going to be deployed is dominated by uncertainty. Machine learning techniques offer flexibility as they can make decisions based on data also in presence of uncertainty. Such techniques have been increasingly used in autonomous vehicles. However, the flexibility offered by such techniques has to balance with the lack of control from the developer of the software. Many challenges need to be overcome before such systems can be trusted to act safely. Generally, we can group our research under the umbrella of the following research question:

RQ2: How to build trustworthy autonomous systems also in presence of uncertainty?

RQ2.1 How to make sure that the resulting system satisfies certain desired properties?

RQ2.2 How to ensure that the resulting system reflects the designer's intentions without manifesting unwanted behaviours?

RQ2.3 How to provide assurances when several autonomous systems collaborate to create a new and more complex system?

Research question **RQ2** has a wide scope and aims to build a trustworthy system also when not all the behaviours are defined at design-time. We have investigated how to use machine learning techniques, such as reinforcement learning, to let the system acquire the desired behaviour at runtime in order to achieve a goal. We have split this problem into two sub-questions. The first one (**RQ2.1**) goes in the direction of formally *verifying* that the behaviour emerging from the system is compliant with some important safety-properties (*invariants*) also while it is exploring the environment. We aim to achieve this challenging goal by using a combination of machine learning and runtime monitoring as described in Paper D and validated in Paper E.

The question **RQ2.2** builds in the direction of assuring that the goals as intended by the designer are correctly transferred to the AS itself. In Paper B, we deal with the problem of conveying the functional requirements to a machine learning agent solely using the reward function. A problem with such systems is that the agent could manifest unwanted behaviours and consequences. We aim to give the system designer more control and understanding of the reward function on which the reinforcement learning agent is going to build its decision making policy.

Finally, the research question **RQ2.3** does not focus on the behaviour of a single autonomous system but instead, it deals with a system constituted by several individual systems (System of Systems, SoS). We want to investigate how current verification techniques can be applied to such systems. We have addressed this research question in Paper C by modelling a SoS scenario where multiple vehicles have to interact in order to form a platoon. The individual systems can independently adapt their behaviour at runtime and the rules defining the SoS are modelled at design-time. We have modelled different modes where the SoS can be and defined properties we want the SoS to have as *invariants*. Finally, we have formally verified that in randomly generated scenarios the resulting system is compliant with them using model checking. With respect to Figure 1.2, this work concerns the elicitation and verification of system's invariants.

Methodology: For **RQ2** we have used *Design Science* as research methodology [40]. The two main activities are the of design science are: (i) designing an artefact, (ii) empirically evaluating its performance in a context. In most of the papers addressing RQ2 the artefact is the proposed method. In Paper E and Paper B we have implemented a framework that supports the proposed methods; the validation has been done by defining a case study and collecting data from experiments conducted by *computer simulation* [41]. In Paper C the artefact is the protocol for vehicle platooning and it has been validated by formal-verification (model-checking) on randomly generated scenarios. Finally, in Paper D we have proposed a general method, a specific case of this method is then implemented and and validated in Paper E.

1.4 Summary of Studies

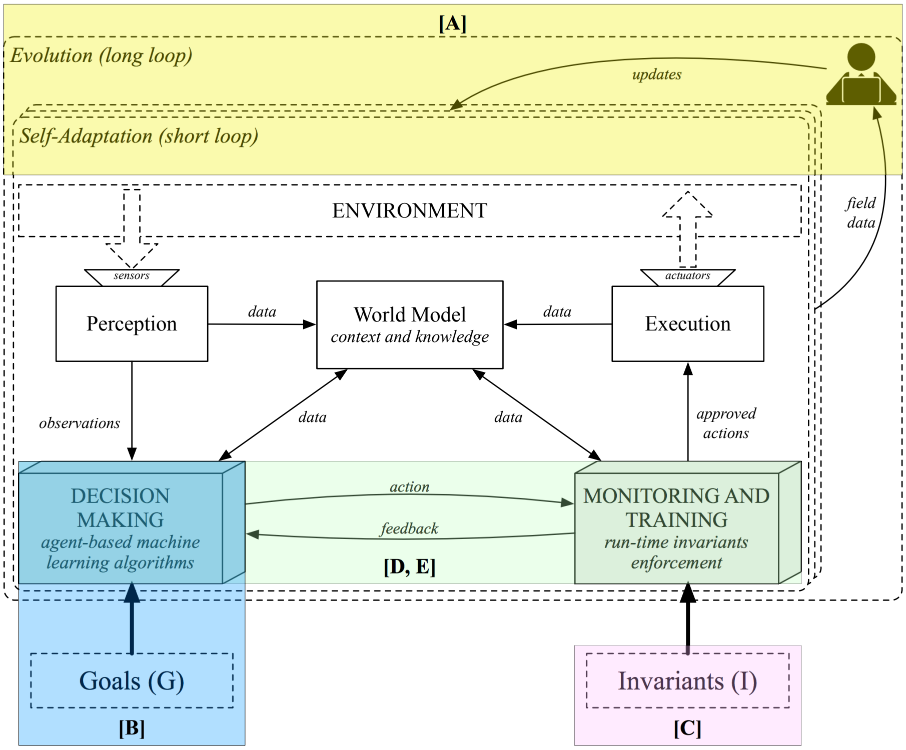

Figure 1.2: Proposed Framework for Trustworthy Self-Adaptive Autonomous System. The letters indicate the papers contributing in different parts of the framework.

Figure 1.2 maps the contributions of this thesis into a general framework describing our proposal of Trustworthy Self-Adaptive Autonomous System. This framework has been first introduced in [42], where we describe a method that combines machine learning for decision-making with assurance techniques to guarantee the preservation of safety-critical system's invariants. On one level - short loop - each software agent locally self-adapts at runtime using machine learning techniques. On another level - long loop - field data collected by a set of agents is exploited to produce new versions of the software to be then deployed on the single autonomous systems.

The decisions are driven by the *goals* that the agent must achieve, depending on the observation perceived from the environment and the current state. On one side, the agent uses *reinforcement learning* to select the actions to be performed. On the other side, important properties, such as safety of requirement, are broken down in terms of *invariants* that must be assured at all times. *Monitor and training* will prevent the agent to choose an action that violates the invariants and it will also train the agent to perform better in the future.

In the following sections we describe the included papers starting from the first research questions **RQ1** addressing the state of the art (Paper A), then going to **RQ2** presenting the papers about achieving goals (Paper B), preserving the invariants (Paper C) and finally monitoring the behaviour of the AS at runtime (Paper D and Paper E).

1.4.1 State of art, future trends, and challenges

During recent years, the automotive sector has been experiencing major transformations. Connected vehicles will benefit from Intelligent Transport Systems (ITS), Smart Cities, and the Internet of Things (IoT). They will combine data from inside vehicle with external data coming from the environment (other vehicles, the road, the signs, and the cloud). Software is the main enabler of autonomous vehicles and its volume has been keeping increasing for years; it is expected to increase by 50% by 2020 [43]. 80% to 90% of the innovation within the automotive industry is based on electronics, and a big part of electronics is software [44, 45].

Paper A deals with Autonomous Systems in a broader scope, identifying what are the challenges and possible solutions in the automotive domain. We describe the different levels of autonomy, from vehicles with partial autonomy to fully autonomous systems that have to continue operating safely also during faults. Several functionalities contribute to increasing the autonomy level of modern vehicles, from assisting the driver during parking manoeuvres to enhancing the vision during the night. Furthermore, the software architecture of such autonomous vehicles has to change in order to accommodate different modules dealing with observations, sensor information, interpretation of the data, and actuation.

In this paper, we report the state of the art, future trends, and challenges of autonomous vehicles, with a special focus on software. We describe some of the functionalities of autonomous vehicles and their software architecture. Then, we have identified some of the key challenges of autonomous vehicles. The use of machine learning is one of the major challenges as the vehicles will have to deal with uncertainties that characterise the environments in which autonomous vehicles will need to operate while guaranteeing safety properties in all situations.

1.4.2 Conveying the right goals to the system

An intelligent autonomous system must figure out how to achieve its goals by itself. The system designer job is to specify *what* goal to achieve and the decision-making component of an Autonomous System must figure out *how* to achieve it. In reinforcement learning the only way that the system designer has to convey the goal to the agent is through the *reward function*. The encoding of system goals into a reward function can lead to unexpected behaviours in the agent, either because the designer does not include or have the correct information or because she/he makes mistakes during the design of the reward function.

Paper B addresses some of the challenges identified in **Paper A** regarding the transfer of *goals* to a reinforcement learning agent, as also shown in Figure 1.2. We have proposed an approach for engineering complex reward functions that can be formally verified against defined properties at design-time and automatically enforced to the agent at runtime. Our aim is to reduce the gap between the designer's intention and the reward specification. By embedding more domain knowledge in the reward function one could avoid the reward hacking phenomenon; this would also help the agent to learn the desired policy faster [46]. However, as reward functions become more complex, in turn, it becomes harder to spot mistakes and to be confident whether the reward values that are finally sent to the agent actually reflect the designer's intentions.

Our work goes in the direction of a better structuring of the reward function with the aim of closing the gap between the designer informal goals and the reward signal. Our contribution is the design and validation of a software infrastructure that enables the verification and enforcement of reward functions to an RL agent. From a high-level perspective our approach, which we have called MoVEMo, consists of four steps:

1. *Modelling* complex reward functions as a network of state machines.

2. *Formally verifying* the correctness of the reward model.

3. *Enforcing* the reward model to the agent at runtime using a monitoring and enforcing approach called Larva.

4. *Monitoring* the behaviour of the agent as it transverses the state machines to collect the rewards.

Steps 1 and 2 are performed by the designer that iterates the reward function model until it is compliant with the high-level properties that she/he expresses. Steps 3 and 4 are automatically derived and performed from the reward model. We have validated our approach in the context of self-driving cars with an open-source driving simulator.

1.4.3 Verifying Invariants

Some system requirements can be expressed in terms of *invariants*, meaning properties that always have to hold, despite the system adaptations. Such properties can target an individual system of multiple systems and the way they interact with each other. Autonomous Systems can also collaborate with other systems forming a System of System (SoS). Due to the complexity of such systems, it becomes hard to verify the correctness of their actions.

Formal verification techniques such as model checking can prove that a system satisfies certain desired formal properties. In **Paper C**, we model an Autonomous Systems as a network *timed-automata* and verify that certain properties hold when they interact. Specifically, we have modelled invariants in terms of temporal logic properties and formally verified such invariants in a system of systems scenario. We have successfully verified several invariants on a vehicle platooning protocol with randomly generated scenarios.

1.4.4 Keep intelligence under control

Using techniques such as *reinforcement learning* we can create systems that autonomously learn which action to execute in order to achieve the desired *informal* goal. When a change occurs, machine learning techniques allow the system to autonomously learn new policies and strategies for actions execution. This flexibility comes at a cost: the developer has no longer full control on the system behaviour. To overcome this issue, we believe that machine learning techniques should be combined with suitable reasoning mechanisms aimed at assuring that the decisions taken by the machine learning algorithm do not violate safety-critical requirements.

Paper D closes the loop in Figure 1.2 describing how to combine the decision-making agent with the assurance of safety-critical properties. The approach aims at creating systems that, on one hand, are able to learn and adapt their behaviour based on changes that occur in the environment using *reinforcement learning*, on the other are able to ensure that adaptation does not cause invariants violation using *runtime monitoring*. The runtime monitoring part is explained in details and evaluated in PAPER E.

1.4.5 Runtime monitoring of the intelligent system

In **Paper E**, we propose a concrete solution to the approach proposed by **Paper D** showing its effectiveness through a large experimentation. This approach, named WISEML, uses reinforcement learning in combination with runtime monitoring to prevent the agent from performing catastrophic actions in the environment. The approach is general and external to the RL algorithm, so it does not modify how the RL algorithm works; in this sense, the approach is agnostic to the RL algorithm since one could use any RL algorithm. WISEML wraps the RL agent at its interfaces and it places it inside a *safety envelope* that protects it from performing actions that violate its safety-critical requirements.

The requirements of the RL agent are expressed in terms of *patterns*, they describe the safety-properties to be enforced by the monitoring component. We have implemented four patterns that one can use to model the invariants: *absence*, *globally*, *precedence* and *response*. Our results show that the runtime monitors will always prevent the agent from violating any of the modelled properties. Furthermore, the RL agent will converge faster to its goals thanks to *reward shaping* that steer the agent towards its goal by modifying its rewards at runtime, according to the compliance or the violation of the monitored properties.

1.5 Conclusions and Future Work

Collecting data at runtime, self-adapt, and continuously evolve are fascinating concepts that pose some serious challenges. When dealing with safety-critical systems, any change to the system must be certified as *safe* before it can be applied. With the intensive use of machine learning techniques, it is hard to test the software and be sure that it will act always in a correct way. Reinforcement learning can be used as a technique to drive the system's self-adaptation and learn how to perform decisions in previously unknown environments. However, the resulting system still has to meet its safety requirements despite the adaptations.

In this thesis, we have presented the state-of-the-art and the challenges of engineering trustworthy autonomous systems. We have proposed and validated methods that go in the directions of combining formal methods with machine learning approaches. Our goal is to engineering systems that continuously adapt using Reinforcement Learning to perform decisions. At the same time, we want to keep the preservation of the system's invariants by continuously monitoring how the system reacts to the changes in the environment. This is performed by a continuous runtime monitoring of the machine-learning generated actions.

In the future, we will focus on the integration of different state-of-the-art methods with the vision to build an Autonomous System that can make safety-certifiable decisions.