Towards a Framework for Certification of Reliable Autonomous Systems

Fisher, Michael; Mascardi, Viviana; Rozier, Kristin Yvonne; Schlingloff, Bernd-Holger; Winikof, Michael ;
Yorke-Smith, Neil

DOI
10.1007/s10458-020-09487-2

Publication date
2021

Document Version
Final published version

Published in
Autonomous Agents and Multi-Agent Systems

Autonomous Agents and Multi-Agent Systems (2021) 35:8
https://doi.org/10.1007/s10458-020-09487-2

Towards a framework for certification of reliable autonomous systems

Michael Fisher[1] · Viviana Mascardi[2] · Kristin Yvonne Rozier[3] · Bernd-Holger Schlingloff[4] · Michael Winikoff[5] · Neil Yorke-Smith[6]

Accepted: 15 November 2020

Abstract

A computational system is called *autonomous* if it is able to make its own decisions, or take its own actions, without human supervision or control. The capability and spread of such systems have reached the point where they are beginning to touch much of everyday life. However, regulators grapple with how to deal with autonomous systems, for example how could we certify an Unmanned Aerial System for autonomous use in civilian airspace? We here analyse what is needed in order to provide verified reliable behaviour of an autonomous system, analyse what can be done as the state-of-the-art in automated verification, and propose a roadmap towards developing regulatory guidelines, including articulating challenges to researchers, to engineers, and to regulators. Case studies in seven distinct domains illustrate the article.

Michael Fisher was at the University of Liverpool when part of this work was done, and Michael Winikoff was at the University of Otago.

✉ Neil Yorke-Smith
n.yorke-smith@tudelft.nl

Michael Fisher
michael.fisher@manchester.ac.uk

Viviana Mascardi
viviana.mascardi@unige.it

Kristin Yvonne Rozier
kyrozier@iastate.edu

Bernd-Holger Schlingloff
hs@informatik.hu-berlin.de

Michael Winikoff
michael.winikoff@vuw.ac.nz

[1] University of Manchester, Manchester, United Kingdom

[2] University of Genova, Genova, Italy

[3] Iowa State University, Ames, IA, USA

[4] Humboldt University and Fraunhofer FOKUS, Berlin, Germany

[5] Victoria University of Wellington, Wellington, New Zealand

[6] Delft University of Technology, Delft, The Netherlands

Published online: 23 December 2020

Keywords Autonomous systems · Certification · Verification · Artificial intelligence

1 Introduction

Since the dawn of human history, humans have designed, implemented and adopted tools to make it easier to perform tasks, often improving efficiency, safety, or security. Indeed, recent studies show a direct relationship between increasing technological complexity, cognitive evolution, and cultural variation [231].

When such tools were simple, the person using the tool had full control over the way the tool should be operated, understood why it worked in that way, knew how the tool should be used to comply with existing rules, and when such rules might be broken if the situation demanded an exceptional use of the tool. For example, our early ancestors could use a hammer for building artefacts, knew why the hammer could be used for their purposes, followed the rules of not using it as a weapon against other humans, but might have chosen to break this rule if their families were in danger (Fig. 1).

However, as tools became more complex and developed into systems composed of many different parts, users lost their broad view on how the system, or even some of its components, worked and – without that know-how – they lost part of their control over the system. But users still retained the capability of using systems following the rules, and breaking the rules if needed. By delegating the control of some basic tasks to the system itself, users gained in efficiency at the expense of exhaustive control (Fig. 2).

Nowadays, the sophisticated systems that we rely on have become so complex that our awareness of *what* actually happens when we exploit some of their functionality is often close to zero. For example, how many people know how a cloud storage system works? Or the complex link between a vehicle's brake pedal and the vehicle speed? Even if we are domain experts, we barely know the complete event/data flow initiated by just pressing one button. This is even more true with the rise of auto-* and self-* systems (auto-pilots, self-driving cars, self-configuring industrial equipment, etc). We therefore can no longer just delegate the control of basic operations. If we want a car to drive by itself, we must also delegate to it the requirement to follow the road traffic rules (Fig. 3).

So far, however, a self-driving car is neither designed nor expected to make decisions in moral-ethical situations [31]. When ethics, and even merely outside-of-scope situations,

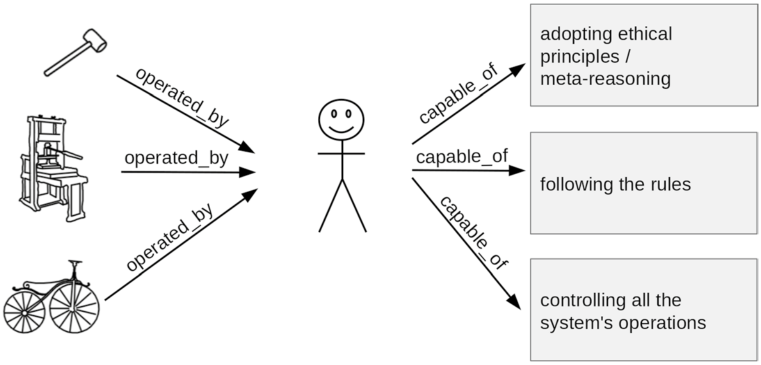

operated_by
operated_by
operated_by
capable_of
capable_of
capable_of
adopting ethical principles / meta-reasoning
following the rules
controlling all the system's operations

Fig. 1 Human user has the full control of the tool

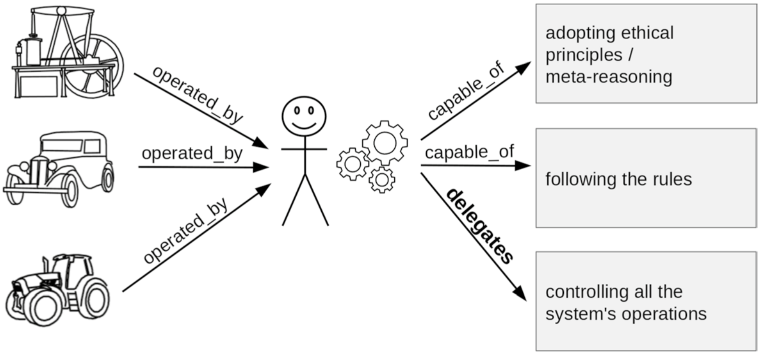

operated_by
operated_by
operated_by
capable_of
capable_of
delegates
adopting ethical principles / meta-reasoning
following the rules
controlling all the system's operations

Fig. 2 Human (partially) delegates the control of the operations

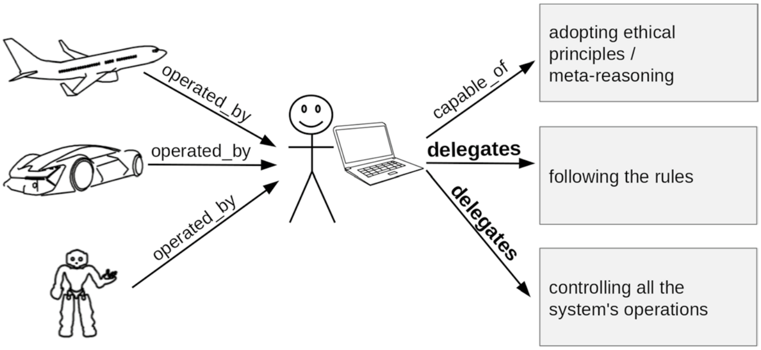

operated_by
operated_by
operated_by
capable_of
delegates
delegates
adopting ethical principles / meta-reasoning
following the rules
controlling all the system's operations

Fig. 3 Human (partially) delegates the respect of the rules

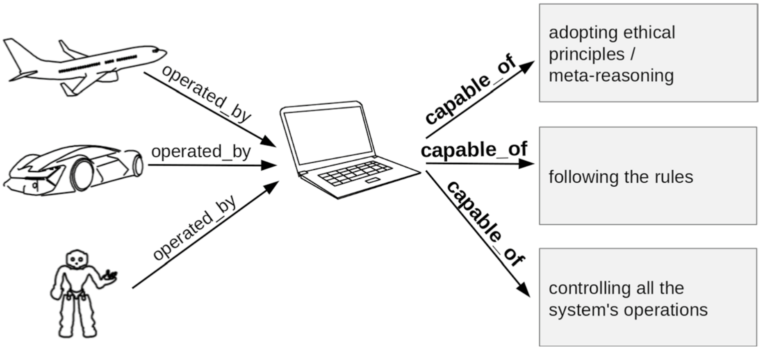

operated_by
operated_by
operated_by
capable_of
capable_of
capable_of
adopting ethical principles / meta-reasoning
following the rules
controlling all the system's operations

Fig. 4 Human replaced by an autonomous software system

bear upon autonomous operation, the human must still be responsible [249]. As an example, if a self-driving car has a mechanical/software failure in a dangerous situation or if it encounters a safety dilemma, responsibility is transferred to the human.

Nevertheless, due to the delegation of more and more capabilities from humans to machines, the scenario depicted in Fig. 4 – where the human is replaced by an autonomous system – is becoming more realistic. This scenario of full autonomy raises many ethical, legal, social, methodological, and technical issues. In this article we address the crucial question: *"How can the reliability of such an autonomous software system be certified?"*

1.1 Terminology

Before exploring this challenging question, we need to define the terminology used in the sequel. By 'we' this article means the authors. When we want to indicate some more general class of individuals, such as 'the scientific community', or 'humankind', we will explicitly use those terms.

We start with *reliability*. The term 'reliable' means "suitable or fit to be relied on" [186]. For systems offering a service or function, reliability means that the service or function is available when needed. A software system is reliable to the extent that it meets its requirements consistently, namely that it makes good decisions in all situations. In some situations, a good decision is simply one that follows given rules, for instance, choosing to stop at a red traffic light. However, in other, hopefully rare situations, rules may need to be overridden, for instance, temporarily driving on the wrong side of the road to avoid a crash.

Answering the question of what constitutes a *good* decision is out of the scope of this article. Ethical decision making has been widely studied by psychologists and philosophers such as Lawrence Kohlberg, who developed the theory of stages of moral development [167–169], and different cultures have a different attitude towards the notion of a *good* decision. Our contribution is not on the philosophical challenges of making a good decision, but on the technological ones.

Reliability is often associated with the notion of a *certification*, "a proof or a document proving that someone is qualified for a particular job, or that something is of good quality" [47]; besides the document, certification also refers to "the process of giving official or legal approval to a person, company, product, etc, that has reached a particular standard" [47]. Human professionals can be certified, and the idea is not new: guilds of arts and crafts were born in the 12th century in many European cities, to regulate and protect the activities of those belonging to the same professional category [161]. Being part of a guild was a certification of the craftman's or merchant's professional skills. As soon as machines partially or completely supplemented professionals, the need to certify machines arose – at least in terms of safety, if not functionality; this is also true of software. Certification of software reliability is a lively research area in software engineering, as we discuss in Sect. 2.2.

We define a system to be *autonomous* if it can make its own decisions and act on them, without external (human) supervision and control. For example, a mobile robot can be completely remote-controlled, in which case it is not autonomous, or it can have a built-in control unit that decides on its moves, such that it becomes *semi-autonomous*. Of course, the boundary separating fully autonomous from non-autonomous systems is not black and white. For example, the robot may be allowed some degree of autonomy, e.g., in path planning, whereas the overall movement goal is imposed by some remote controller.

Definition 1 The levels of autonomy that we will use to classify examples of systems from different domains in Sect. 4, roughly follow the six-grade scale given for autonomous road vehicles by SAE International [221], though, e.g., that standard does not include our *low* layer:

- **No autonomy** The operator is responsible for all tasks that are necessary for allowing the system to provide the functions – and hence achieve the goals – for which it was built.
- **Low autonomy** Straightforward (but non-trivial) tasks are done entirely autonomously (no human poised to take over operation).
- **Assistance systems** The operator is assisted by automated systems, but either remains in control to some extent or must be ready to take back control at any time.
- **Partial autonomy** The automated system takes full control of the system, but the operator must remain engaged, monitor the operation, and be prepared to intervene immediately.
- **Conditional autonomy** The automated system has full control of the operation during specified tasks; the operator can safely turn their attention away but must still be prepared to intervene upon request.
- **High autonomy** The automated system is capable of performing all planned functions under certain circumstances (e.g., within a certain area); the operator may safely leave the system alone.
- **Full autonomy** The system can perform all its intended tasks on its own, no human intervention is required at any time.

Although uninteresting for the focus of this article, we include a *no autonomy* level in our scale to describe those systems that must be entirely operated by a human being to achieve their goals. Consider for example a simple food immersion blender with three manually-selected speeds. This system has no autonomy at all, as it provides the functionality of blending, but the tasks (or *responsibilities*) of selecting the right speed, and of pressing the on/off button to ensure that the food is blended but not warmed, and that blades are not damaged, are entirely controlled by the user. A more sophisticated version of the same blender might have temperature sensors that decrease the speed when the food to be blended is becoming too warm, and switch the power off when the blades meet some unexpectedly hard object. The latter would address the 'not warming food, not damaging blades' goals with some *low*, but not zero, autonomy level.

In addition to defining the *level* of autonomy, we also consider the *scope* of autonomy. This is the level of functionality of the system's autonomous capabilities. For example, one vacuum cleaner might have autonomous capabilities that only encompass traversing a space and avoiding obstacles, while another, more sophisticated model, may also be able to schedule its cleaning to avoid disruption to the human's schedule. We would say that the second model has greater scope of autonomy. The scope and level of autonomy can sometimes be a trade-off: increasing the scope may involve the system doing things that it cannot do fully autonomously, whereas a system with more limited scope may be able to have higher autonomy.

We are particularly interested in fully autonomous systems that can also make their own decisions on *safety-critical* actions, namely actions whose failure could result in loss of life, significant property damage or damage to the environment.[1] Additionally, autonomous systems are often characterised by the need to balance pursuing objectives over a long time period (being *proactive*), with responding to environmental and system changes (being *reactive*).

[1] Adapted from the definition of 'Safety Critical System' [166].

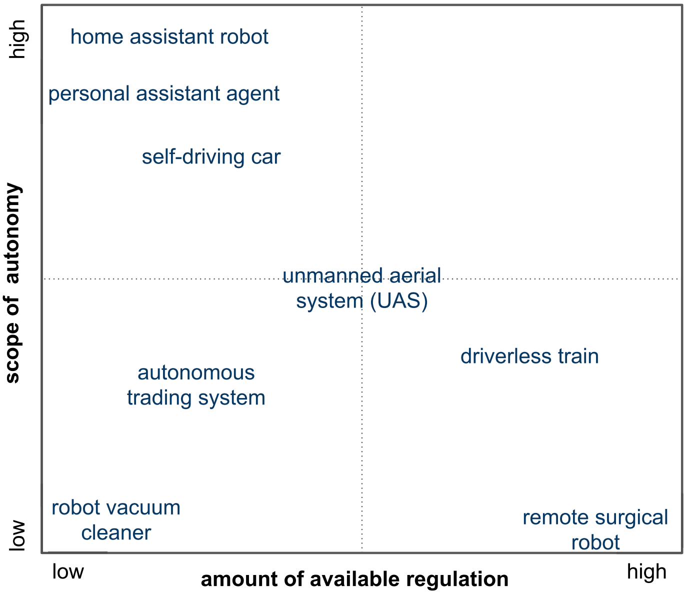


Fig. 5 Comparing domains of autonomous systems in terms of the level of autonomy expected in the (near) future, and the amount of regulation existing today. Note that the level of (expected) autonomy and of existing regulation cannot be precisely measured on a scale. This figure should be interpreted as being roughly indicative

In the sequel, we will also make strong use of the notion of an *agent*. An *autonomous software agent* (agent for short) is an autonomous software system that captures the ability to decide or act independently, while also balancing between being proactive and reactive. We use the standard definition of a *multiagent system* as a system that includes multiple such agents, which may interact in various ways (e.g., communicating using messages or via the environment): see the seminal works [160, 254, 255]. Finally, we consider *rational agents* as those that are structured in terms of *intentional* concepts, such as goals, beliefs, and intentions (synonymously, the terms ‘cognitive agent’ or ‘intelligent agent’ sometimes appear in the literature [255]).

Figure 5 compares different domains of autonomous systems in terms of the expected autonomy and available regulation. Although the scope[2] of (expected) autonomy and the level of regulation cannot be measured precisely, the figure highlights that there are systems (top left quadrant) with considerable scope for autonomy, but limited available regulation. These are the sorts of systems that particularly require work to be able to shift them across to the right by increasing the available regulation. We discuss each of these domains in Sect. 4, with the exception of remote surgical robots, since there is not enough autonomy permitted to such systems.

[2] We use the scope of autonomy in this figure, rather than the level of autonomy, because for the systems considered there is a trade-off: the systems vary in the scope of autonomy, but for many of these systems the scope is set (by designers) in order to allow the system to display high levels of autonomy, making scope of autonomy a more useful differentiator than the level of autonomy.

It is worth noting that although many systems can be viewed as being rational agents, we only do so when there is benefit in adopting an intentional stance and viewing the system in these terms. For example, a thermostat makes decisions, and we could ascribe it a goal to keep the room at a fixed temperature. However, the behaviour of a thermostat is simple enough that there is no benefit to viewing it in terms of goals and beliefs [64].

It is important to highlight that, for purposes of certification, or other regulatory procedures, we sometimes need to consider not just *what* a software system did, but also *why* it did it. For instance, there is a difference between a car breaking the speed limit because it has an incorrect belief about the speed limit, and a car going too fast because it believes that temporarily speeding is the best, or even the only, way to avoid an accident.[3]

1.2 Audience, contributions, and structure

This article assesses what is needed in order to provide verified, reliable behaviour of an autonomous system, analyses what can be done as the state of the art in automated verification, and proposes a roadmap towards developing certification and broader regulation guidelines.

This article thus has three audiences. Firstly, we address regulators, who might find the proposed roadmap useful as a path towards being able to meaningfully regulate these sorts of systems. Secondly, engineers and developers who develop such systems might find it useful in seeing how/where these systems need greater analysis. Thirdly, academic researchers can advance the state of the art by finding better ways of dealing with the challenges that we articulate.

We advance the literature by:

1. proposing a framework for viewing (and indeed building) autonomous systems in terms of three layers;
2. showing that this framework is general, by illustrating its application to a range of systems, in a range of domains;
3. discussing how certification/regulation might be achieved, breaking it down by the three layers; and
4. articulating a range of challenges and future work, including challenges to regulators, to researchers, and to developers.

The remainder of the article is structured as follows. Section 2 reviews the current situation in terms of regulation and certification of (semi-)autonomous systems, and of the issues still open. Section 3 assesses what could be done in the near future; it develops our three-layer reference framework, discusses what we need from regulators, proposes a process for deriving verification properties, and reviews in more detail verificaton techniques. Section 4 discusses a set of case studies in different application domains. Section 5 looks at challenges in research, engineering, and regulation. Section 6 summarises and indicates future directions.

[3] Here we see the interplay between norms of different types [63]: current jurisdiction in Germany at the time of writing is that one is not allowed to transgress speed limits even in life-threatening situations. The argument is that even in order to save a human life one is not supposed to endanger another one.

2 Looking back

All systems, be they autonomous or not, that operate in a human society need to conform to some legal requirements. These legal requirements may be generic and apply to all products, or specific. Often these requirements are based on *regulations*, that we define as: rules, policies, and laws set out by some acknowledged authority to ensure the safe design and operation of systems.[4]

Relating to the concept of regulation, in the context of this paper *certification* can be specified as: the determination by an independent body that checks whether the systems are in conformity or compliant with the above regulations. Certification involves a legal, rather than scientific, assessment and usually appeals to *external review*, typically by some *regulator*.

The certification processes, and hence regulators, in turn appeal to *standards*, namely documents (usually produced by a panel of experts) providing guidance on the proving of compliance.

There are a plethora of different standards, issued by a wide range of different standardisation organisations. Amongst the most well known are *CENELEC* [79], *IEC* [130], *IEEE* [113], and *ISO* [139], to name just a few. Many of these organisations provide generic standards relevant across many (autonomous) system domains. For particular sectors, the regulatory bodies – and there may be several for each sector – have a range of specific standards. In Sect. 2.1 we present some of the most relevant existing standards, and in Sect. 2.2 we overview some methods and tools suitable for certification of software systems. It is important to note, however, that nowadays automatically moving from written standards to formal specifications that can be fed to tools able to check, verify, and certify the system's behaviour, is not possible. Also, most existing standards say little, if anything, about *autonomy* and *uncertainty*, the situation where autonomy is more needed, but also more dangerous. Nevertheless, they prescribe important properties with which systems should aim to comply. Sect. 2.3 faces some issues raised by autonomous systems, which are not (yet) satisfactorily addressed by current standards and regulations, including how we might link together the achievements described in the first two sections, and how we might deal with autonomy and uncertainty.

2.1 Standards

Tables 1 to 5 present some standards grouped by domains where autonomy potentially plays a crucial role. The most sophisticated illustrative examples in Sect. 4 arise from these domains. We do not claim this to be either exhaustive or systematic: this section is only meant to give the reader an idea of the complexity and wide variety of existing standards by providing examples issued by different organisations. It is important to note that there is a *vast* array of standards, many of which are never used by any regulator.

Table 1 illustrates some standards in the robotics domain. Most of them come from ISO. A Technical Committee of the ISO created in 2015 [148] is in charge of the standardisation of different robotics fields, excluding toys and military applications. In 2015, IEEE

[4] The definition provided by the Cambridge English Dictionary is "the rules or systems that are used by a person or organisation to control an activity or process" [49]. We customise this definition for systems that may perform safety-critical actions.

Table 1 Examples of standards for robotics

Code	Title	Year	Abstract
ISO 13482 [147]	Robots and robotic devices – Safety requirements for personal care robots	2014	Requirements and guidelines for the inherently safe design, protective measures, and information for use of personal care robots, in particular mobile servant robots, physical assistant robots, person carrier robots.
IEEE 1872 [115]	IEEE standard ontologies for robotics and automation	2015	A core ontology that specifies the main, most general concepts, relations, and axioms of robotics and automation, intended as a reference for knowledge representation and reasoning in robots.
ISO/TS 15066 [154]	Robots and robotic devices – collaborative robots	2016	Safety requirements for collaborative industrial robot systems and the work environment, supplementing the requirements and guidance on collaborative industrial robot operation given in ISO 10218-1 and ISO 10218-2.
ISO/TR 20218-1, ISO/TR 20218-2 [155, 157]	Robotics – safety design for industrial robot systems – Part 1 (End-effectors) & Part 2 (Manual load/unload stations)	2017, 2018	Applicable to robot systems for manual load/unload applications in which a hazard zone is safeguarded by preventing access to it, and both access restrictions to hazard zones and ergonomically suitable work places must be considered. Guidance on safety measures for the design and integration of end-effectors used for robot systems.
ISO/TR 23482-2 [156]	Robotics – application of ISO 13482 – Part 2: application guidelines	2019	Guidance on the use of ISO 13482 to facilitate the design of personal care robots in conformity with ISO 13482, including new terms and safety requirements introduced to allow close human-robot interaction and human-robot contact in personal care robot applications.

Table 2 Example of standards for medical-assistive technologies

Code	Title	Year	Abstract
IEC/TR 60601-4-1 [137]	Medical electrical equipment – Part 4-1: Guidance and interpretation	2017	Guidance to detailed risk management and usability engineering processes for medical electrical equipment (MEE) or medical electrical systems (MES), employing a degree of autonomy (DOA), and guidance on considerations of basic safety and essential performance for MEE and MES with a DOA.

developed an ontology for agreeing on a shared terminology in robotics, and delivered it as a standard.

Table 2 summarises some facts of one IEC standard dealing with medical equipment. Many standards in this domain exist, also delivered by ISO, which issued more than 400 standards focusing on health [152] thanks to three Technical Committees dealing with medical equipment [141–143] and one dealing with health informatics [144]. We selected [137] as an example from the medical technologies domain, because it focuses on equipments with ‘a degree of autonomy’.

Nearly 900 ISO standards have been developed for the automotive sector [149]. One of the influential is the ISO 26262 [151], born as an adaptation of the Functional Safety standard IEC 61508 for Automotive Electric/Electronic Systems [132]. Published in 12 individual parts, ISO 26262 was updated in 2018 to keep abreast of today’s new and rapidly evolving technologies, and be relevant to even more applications. IEEE is also developing standards in the automotive sector, ranging from public safety in transportation-related events [114] to system image quality [116]. More than three dozen IEC technical committees and subcommittees cover the standardisation of equipment used in, and related to, road vehicles. As an example, the IEC TC 69 [135] is preparing international standards for road vehicles totally or partly electrically propelled from self-contained power sources, and for electric industrial trucks. Table 3 presents one standard for each of the three organisations above, ISO, IEEE, and IEC.

Compared to other domains, the railway homologation and operation is strictly regulated. The IEC Technical Committee 9 [136] is responsible for international standardisation of electrical equipment and systems used in railways. The ISO Technical Committee 269 [146] complements IEC TC 9 by addressing the standardisation of all systems, products, and services specifically related to the railway sector, not already covered by IEC TC 9. Both work closely with the International Union of Railways (UIC, [158]) and the International Association of Public Transport (UITP, [126]). Through the CENELEC 50128 standard [51], CENELEC assesses the conformity of software for use in railway control that may impact safety, i.e., software whose failures can affect safety functions. Table 4 exemplifies standards in the railway sector by presenting one standard from ISO dealing with project management; one series from IEC dealing with reliability, availability, maintainability, and safety; and the CENELEC 50128 standard.

The quantity of existing standards in the aerospace domain is huge. Established in 1947, ISO/TC 20 [140] is one of the oldest and most prolific ISO technical committees. IEEE has published nearly 60 standards dealing with aerospace electronics, and IEC has two Technical Committees dealing with avionics-related issues [128, 129]: these committees developed about 30 standards. Other relevant standards bodies must be mentioned as well. The mission of the European Union Aviation Safety Agency (EASA, [81]) is to ensure the highest common level of safety protection for EU citizens and of environmental protection; to provide a single regulatory and certification process among Member States; to facilitate the internal aviation single market and create a level playing field; and to work with other international aviation organisations and regulators. The US Federal Aviation Administration (FAA, [85]) summarises its mission as “to provide the safest, most efficient aerospace system in the world.” Finally, the US Radio Technical Commission for Aeronautics (RTCA, [205]) aims at being “the premier public-private partnership venue for developing consensus among diverse and competing interests on resolutions critical to aviation modernisation issues in an increasingly global enterprise.” In Table 5 we present standards from EASA, FAA, and RTCA, including two standards dealing with Unmanned Aircraft Systems and drones.

Table 3 Examples of standards in the automotive domain

Code	Title	Year	Abstract
IEEE-P2020 [116]	Standard for automotive system image quality	2016	This standard addresses the fundamental attributes that contribute to image quality for automotive Advanced Driver Assistance Systems applications, as well as identifying existing metrics and other useful information relating to these attributes.
ISO 26262 [151]	Road vehicles – functional safety	2018	Safety is one of the key issues in the development of road vehicles. With the trend of increasing technological complexity, software content, and mechatronic implementation, there are increasing risks from systematic failures and random hardware failures. Both are within the scope of functional safety. The ISO 26262 series of standards includes guidance to mitigate these risks by providing appropriate requirements and processes.
IEC 63243 ED1 [138]	Interoperability and safety of dynamic wireless power transfer (WPT) for electric vehicles	2019	This standard, developed by the IEC TC 69, was due for release in 2021. It will specify definitions and conditions of interoperability and safety for magnetic-field dynamic WPT for electric vehicles and the associated safety requirements.

Table 4 Examples of standards in the railway domain

Code	Title	Year	Abstract
IEC 62278 series [131, 133, 134]	Railway applications – specification and demonstration of reliability, availability, maintainability, and safety (RAMS)	2002, 2010, 2016	The documents under the IEC 62278 umbrella provide Railway Authorities and the railway support industry with a process for implementation of a consistent approach to managing reliability, availability, maintainability, and safety (RAMS). The process can be applied systematically by a Railway Authority or the railway support industry, throughout all phases of the life cycle of a railway application, to develop railway-specific RAMS requirements, and to achieve compliance with these requirements.
CENELEC 50128 [51]	Railway applications – communication, signalling, and processing systems – software for railway control and protection systems	2011	Specification of the process and technical requirements for the development of software for programmable electronic systems for use in railway control and protection applications, aimed at use in any area where there are safety implications.
ISO/TR 21245 [150]	Railway applications – Railway project planning process – Guidance on railway project planning	2018	Guidance on railway project planning for decision making, based upon the principles of ISO 21500 [145], by incorporating characteristics specific to railway projects. The document is meant to be used by any type of organisation and be applied to any type of railway project, irrespective of its complexity, size, or duration. It provides neither detailed requirements nor specific processes for certification.

Table 5 Examples of standards for the aerospace sector

Code	Title	Year	Abstract
RTCA DO-254 [208]	Design assurance guidance for airborne electronic hardware	2000	This document is intended to help aircraft manufacturers and the suppliers of aircraft electronic systems assure that electronic airborne equipment safely performs its intended functions. The document also characterises the objectives of the design life cycle processes and offers a means of complying with certification requirements.
RTCA DO-333 [209]	Formal methods supplement to DO-178C and DO-278A	2011	Additions, modifications, and substitutions to DO-178C (see below) and DO-278A [207] objectives when formal methods are used as part of a software life cycle, and the additional guidance required. It discusses those aspects of airworthiness certification that pertain to software production, using formal methods for systems approved using DO-178C.
RTCA DO-178B, DO-178C/ED-12C [206, 210]	Software considerations in airborne systems and equipment certification	2012	Recommendations for producing software for airborne systems and equipment that performs its intended functions with a level of confidence in safety that complies with airworthiness requirements. Compliance with the objectives of DO-178C is the primary means of obtaining approval of software for use in civil aviation products.
FAA Part 107 [87]	Operation and certification of small unmanned aircraft systems	2016	Addition of a new part 107 to Title 14 Code of Federal Regulations [86] to allow for routine civil operation of small Unmanned Aircraft Systems (UAS) in the National Airspace System and to provide safety rules for those operations. The rule limits small UAS to daylight and civil twilight operations with appropriate collision lighting, confined areas of operation, and visual-line-of-sight operations.
Regulation (EU) 2018/1139 [234]	Regulation (EU) 2018/1139 of the European parliament and of the council of 4 July 2018	2018	First EU-wide regulations for civil UAS with a strong focus on the particular risks of the operations. The regulations take into account the expertise of many international players in the UAS domain; they allow remotely-piloted aircraft to fly safely in European airspace and bring legal certainty for this rapidly expanding industry.

Having reviewed relevant standards in various domains, we next turn to briefly reviewing techniques for certification of software systems.

2.2 Certification of traditional software systems

In the late 1980s, with software applications becoming more and more pervasive and safety-critical, many scientists began to address the problem of certifying them. One of the first papers in this research strand was ‘Certifying the Reliability of Software’ [60]. It proposed a certification procedure consisting of executable product increments, representative statistical testing, and a standard estimate of the mean time to failure of the system product at the time it was released. Subsequently, [252] presented a more mature method of certification, consisting of five steps, and addressing certification of both components and full systems: 1) modelling of software usage; 2) derivation of usage profile; 3) generation of test cases; 4) execution of test cases and collection of failure data; and 5) certification of reliability and prediction of future reliability. Due to the infeasibility of quantifying the reliability of life-critical real-time software [46], formal methods emerged as a critical component in software certification. Further, [203]. pointed out that “*if a component has been verified by a mathematical proof of correctness, you may be able to attribute a high degree of reliability to it*.” This paved the way to works where the software certification of safety-critical systems was based on formal methods. It is worth noting that the already-mentioned IEC 61508 standard [132] recommends that *formal methods* be used in software design and development in all but the lowest Safety Integrity Levels.

Among this wide range of work, we mention [103, 104], which achieve certification by annotating the code with preconditions and postconditions, exploiting a five-step process for establishing the property to be verified, and finally demonstrating that the code satisfies the property. In contrast to previous work [26, 248] in the operating systems and database domains, this work addressed the problem of making the verification of security-critical code affordable.

Many mature formal and semi-formal techniques are widely used to certify software: model checking, theorem proving, static analysis, runtime verification, and software testing. We discuss these established techniques in Sect. 3.4 where we also present our vision of the future. The reason is that their adoption is recognised to be crucial for certifying systems that are autonomous, as witnessed, for example, by the recent establishment of the IEEE Technical Committee for Verification of Autonomous Systems in 2019 [125], and by the 2020 FLY AI Report, which explicitly mentions verification/validation as means for ensuring the safe use of AI in aviation [78]. Besides introducing (semi-)formal verification techniques, in Sect. 3.4 we compare them along the five dimensions of inputs, outputs, strengths, weaknesses, and applicability with respect to our reference three-layer framework presented in Sect. 3.1.

As observed in surveys (e.g., [4, 58]), other approaches to software certification have been exploited, besides (semi-)formal methods. Among them, *assurance cases* were proposed as a viable way to certify safety-critical applications [214].

> *An assurance case is an organised argument that a system is acceptable for its intended use with respect to specified concerns (such as safety, security, correctness).*

This analysis of 82 works published between 1994 and 2016 concluded that assurance cases are more widely applied in the areas of transportation, energy, medicine, and military applications [214]. The adoption of assurance cases is rapidly spreading both in academic works and in industry; [65] presents the AdvoCATE toolset for assurance case automation developed by NASA, and overviews more than 20 research and commercial tools suitable for creating structured safety arguments using Claims-Argument-Evidence (CAE) notation [32], and/or Goal Structuring Notation (GSN) diagrams [228]. As a final remark, we observe that software certification is so challenging that the adoption of the same method or tool across different domains is often impossible. Hence, many domain-dependent proposals exist such as for robotics [197], medical systems [13], the automotive sector [14, 260], unmanned aircraft [239, 247], and railway systems [72].

2.3 Open issues in the certification of autonomous systems

Current standards and regulations are not ready for coping with autonomous systems that may raise safety issues, and hence need to undergo a formal process to be certified. One main issue in their adoption is the format in which standards are currently specified: textual descriptions in natural language. The second issue is the lack of consideration, and sometimes even of clear understanding, of the 'autonomy' and 'uncertainty' notions. Sect. 2.3.1 and 2.3.2 discuss these two issues, respectively.

2.3.1 Certifying systems against textual descriptions and system runs

Let us suppose that the approach followed for the certification process is based on verification. Verifying – either statically or dynamically – scientific and technical requirements of complex and autonomous software applications is far from being an easy task but, at least, formalisms, methodologies, and tools for representing and processing such requirements have been studied, designed, and implemented for years, within the formal methods and software engineering communities.

When requirements have a legal or even ethical connotation, such as the standards discussed in Sect. 2.1, their verification may be hard, if not impossible. Such 'legal requirements' are written in natural language: in order to verify that a system complies with them, a step must be made to move from the natural language to a formal, unambiguous one. Creating such specifications is the biggest bottleneck in the verification of autonomous systems [218].

The literature on this topic is vast, but running existing algorithms on existing standards, and expecting to get a clean, consistent, complete formal specification ready to be verified, is currently not realistic. For example, ARSENAL [100] converts natural language requirements to formal models in SAL [24], a formal language for specifying transition systems in a compositional way, and in LTL. Although equipped with a powerful analysis framework based on formal methods, and despite its ability to generate a full formal model directly from text, ARSENAL has documented limitations when dealing, for example, with different ways to express negation and with co-locations like 'write access'. Also, the rules used to feed ARSENAL in case studies so far seem to follow a simple and regular pattern, with 'if' and 'when' conditions clearly defined.

Other works address similar problems in the software engineering research area [62, 204, 261], in the agricultural regulation domain [77], and – up to some extent – in the digital forensics field [258], but the results are far from being applicable to complex, unstructured, heterogeneous standard specifications.

Process mining [243] is an emerging discipline aimed at discovering precise and formal specifications of processes, based on data generated by instances of those processes.

It builds on process model-driven approaches and data mining. There are many ways business processes can be represented using formal languages. Most of them are inspired by Petri Nets [242], but there are also proposals for formalisms based on LTL [180], that could be directly used to feed a model checker or a runtime monitor. However, in order to certify the system, the scientists in charge for the certification process would need:

1. logs of real executions of the system, to mine a precise representation of its functioning (namely, a model of the system's behaviour),
2. properties that the process must verify, either represented in a logical form or translated into a logical form from a natural language description, using the techniques presented above, and
3. a model checker, for checking that the properties are verified by the system's model.

Even if all the three items above were available, the certification would just state that the observed real executions from which the model was extracted, met the properties. Nothing can be stated on all the other not yet observed, but still possible runs of the system. The main challenge raised by this scenario is in fact that, being data-driven, the mined model only covers the already observed situations. It is an approximate specification of how a system behaves in *some* normal operational scenarios meeting the rules, and in *some* scenarios where rules are broken.

At the current state of the art, certifying large and complex (autonomous) systems agains standards based on textual descriptions and system runs is out of reach, and not only because of scientific and technical obstacles: current regulations are indeed not appropriate for autonomous systems. We note the breadth of (mainly academic) work tackling formal methods for (autonomous) robotic systems [178] and would expect this to impact upon regulation and certification in the future, to make them aligned with the developments in the autonomous systems area.

2.3.2 Dealing with autonomy and uncertainty

The standards and regulatory frameworks described in Sect. 2.1 essentially apply to existing systems, but lack some aspects we would expect of future, more complex, and autonomous systems. The first issue is *uncertainty*, the second is *autonomy*. Let us deal with each in turn.

Uncertainty. Current approaches to certification and regulation often assume that:

1. there is a *finite* set of potential hazards/failures,
2. that these can all be identified beforehand, and
3. that this finite set will not change over the lifetime of the system.

If all the above are true then we can use a risk/mitigation based approach since we know what problems can occur.

However, as we move to much more complex environments where we cannot predict every (safety) issue then the above assumptions become problematic. As we provide more AI components, such as online learning modules, we are not only unsure of what the environment will look like but also unsure of what behaviours our system will have (since it might have learnt new ones). All these issues pose severe problems for the current techniques for identifying hazards/faults, assessing risk/mitigation, and building safety-cases.

In more sophisticated systems, such as a domestic robotic assistant with healthcare and social responsibilities, improved ways of regulating such systems will likely have to be constructed. Without such improvements, the existing approaches will impose the above assumptions, stifling application in all but the most static environments.

Autonomy. A second issue is that the concept of 'autonomy' is not well understood in existing standards/regulations. The standards mentioned so far regulate the requirements, behaviour, and development process of complex and sophisticated systems. These systems may show some degree of autonomy, but autonomy is not their most characterising feature, and the standards are neither driven, not strongly influenced, by it. Indeed, the issue of 'autonomy' is conspicuously absent from most existing standards, as well as the ethical issues that it raises. There are only a few, very recent exceptions.

In 2016, the British Standards Institution (BSI, [42]) developed standards on ethical aspects of robotics. The *BS 8611* standard provides a guide to the *Ethical Design and Application of Robots and Robotic Systems* [43]. As stated in its overview:

> *BS 8611 gives guidelines for the identification of potential ethical harm arising from the growing number of robots and autonomous systems being used in everyday life. The standard also provides additional guidelines to eliminate or reduce the risks associated with these ethical hazards to an acceptable level. The standard covers safe design, protective measures, and information for the design and application of robots.*
>
> [...]
>
> *The new standard builds on existing safety requirements for different types of robots, covering industrial, personal care, and medical.*

While the BSI feeds in to ISO standards, the above ethical standard has not yet been adopted by ISO.

In a large, international initiative, the IEEE, through its *Global Initiative on Ethics of Autonomous and Intelligent Systems* [112], has begun to develop a range of standards tackling autonomy, ethical issues, transparency, data privacy, trustworthiness, etc. These standards are still in their early stages of development; Table 6 provides references to those that are more closely related to autonomous systems. The year reported in the table is the Project Authorisation Request (PAR) approval date.

Many efforts in the 'ethics of autonomous systems' research field converged in the *Ethically Aligned Design* document released in 2019 [235]: the document is the result of an open, collaborative, and consensus building approach lead by the IEEE Global Initiative. While not proposing any rigorous standard, it makes recommendations on how to design 'ethics aware' so-called 'autonomous and intelligent systems' (A/IS), and provides reasoned references to the IEEE P70** standards and to the literature.

To give an example, one of the eight general principles leading the A/IS design is *transparency* – the basis of a particular A/IS decision should always be discoverable. The associated recommendation is as follows.

> *A/IS, and especially those with embedded norms, must have a high level of transparency, from traceability in the implementation process, mathematical verifiability of its reasoning, to honesty in appearance-based signals, and intelligibility of the system's operation and decisions.* [235, page 46]

While this represents a very good starting point towards agreeing on which behaviours an A/IS should exhibit, certifying that an A/IS has a high level of transparency, based on the recommendation above, is not possible. Moving from well-known and clear rules

Table 6 Examples of IEEE Standards related to ethics of autonomous systems

Code	Title	PAR Appr.	Abstract
IEEE P7000 [117]	Model process for addressing ethical concerns during system design	2016	Process model by which engineers and technologists can address ethical consideration throughout the various stages of system initiation, analysis, and design.
IEEE P7001 [118]	Transparency of autonomous systems	2016	This standard describes measurable, testable levels of transparency, so that autonomous systems can be objectively assessed and levels of compliance determined.
IEEE P7002 [119]	Data privacy process	2016	Requirements for a systems/software engineering process for privacy-oriented considerations regarding products, services, and systems utilising employee, customer, or other external user's personal data.
IEEE P7003 [120]	Algorithmic bias considerations	2017	Specific methodologies to help users certify how they worked to address and eliminate issues of negative bias in the creation of their algorithms, where 'negative bias' infers the usage of overly subjective or uniformed data sets, or information known to be inconsistent with legislation or with instances of bias against groups not necessarily protected explicitly by legislation.
IEEE P7006 [121]	Standard for personal data artificial intelligence (AI) agent	2017	Technical elements required to create and grant access to a personalised Artificial Intelligence (AI) that will comprise inputs, learning, ethics, rules, and values controlled by individuals.
IEEE P7007 [122]	Ontological standard for ethically driven robotics and automation systems	2017	The standard establishes a set of ontologies with different abstraction levels that contain concepts, definitions, and axioms that are necessary to establish ethically driven methodologies for the design of robots and automation systems.
IEEE P7008 [123]	Standard for ethically driven nudging for robotic, intelligent, and autonomous systems	2017	'Nudges' as exhibited by robotic, intelligent, or autonomous systems are defined as overt or hidden suggestions or manipulations designed to influence the behaviour or emotions of a user. This standard establishes a delineation of typical nudges (currently in use or that could be created).
IEEE P7009 [124]	Standard for fail-safe design of autonomous and semi-autonomous systems	2017	Practical, technical baseline of specific methodologies and tools for the development, implementation, and use of effective fail-safe mechanisms in autonomous and semi-autonomous systems.

written in natural language to their formal counterpart is hard, and formalising recommendations is currently out of reach, as we discuss in Sect. 2.3.1.

3 Ways forward

What is the way forward? There are a number of elements that we can bring together to address and support regulatory development. These span across:

- *architectural/engineering issues* — constructing an autonomous system in such a way that it is amenable to inspection, analysis, and regulatory approval,
- *requirements/specification issues* — capturing exactly how we want our system to behave, and what we expect it to achieve, overcoming the difficulties arising when human-level rules do not already exist, and
- *verification and validation issues* — providing a wide range of techniques, across different levels of formality, that can be used either broadly across the system, or for specific aspects.

This second item is particularly important [218]: if we do not know what is expected of the system, then how can we verify it? In traditional systems, the expected behaviour of the human component in the overall system, be they a pilot, driver, or operator, is often underspecified. There is an assumption that any trained driver/pilot/operator will behave *professionally*, yet this is never spelled out in any system requirement. Then, when we move to autonomous systems, where software takes over some or all of the human's responsibilities, the exact behaviour expected of the software is also under-specified. Consequently, this leads to a requirement for greater precision and level of detail that we require from regulatory authorities and standards.

This section presents an outline for a way forward, covering the three elements. Firstly, a key (novel) feature of our proposed approach is a three-layer framework (Sect. 3.1) that separates dealing with rule-compliant behaviour in 'normal' situations from dealing with abnormal situations where it may be appropriate to violate rules. For example, a system might consider driving on the wrong side of the road if there is an obstacle in its way and it is safe to use the other lane. Secondly, we consider what we need from regulators (Sect. 3.2) and define a process for identifying properties to be verified by considering how humans are licensed and assessed (Sect. 3.3). Thirdly, we review existing verification techniques (Sect. 3.4), including their strengths, weaknesses, and applicability.

3.1 A reference three-layer autonomy framework

In order to distinguish the types of decisions made by autonomous systems, we present a reference three-level framework[5] for autonomy in Fig. 6. This brings together previous work on:

[5] We use 'framework' rather than 'architecture' for two reasons. Firstly, to avoid confusion with an existing (but different) three layer architecture for robots. Secondly, because this framework may not be realised in terms of a software architecture that follows the same three layers.

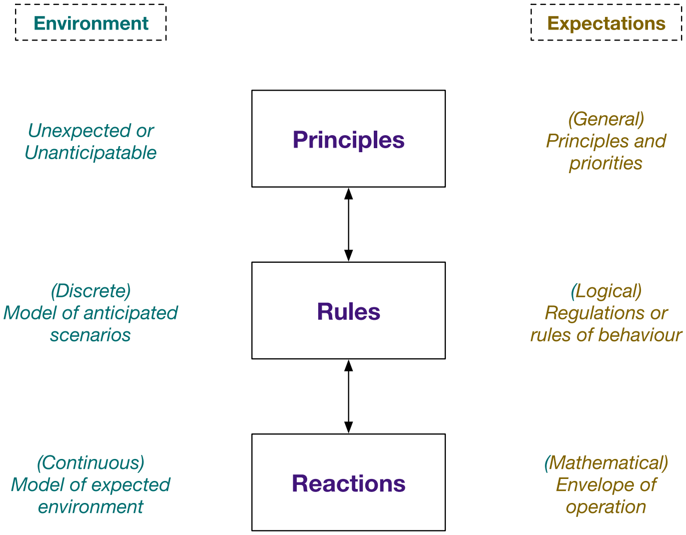

Environment
Expectations
Unexpected or Unanticipatable
Principles
(General) Principles and priorities
(Discrete) Model of anticipated scenarios
Rules
(Logical) Regulations or rules of behaviour
(Continuous) Model of expected environment
Reactions
(Mathematical) Envelope of operation

Fig. 6 A reference three-layer autonomy framework

1. *The separation of high-level control from low-level control in systems architectures.*
   This is a common trend amongst hybrid systems, especially hybrid control systems, whereby discrete decision/control is used to make large (and discrete) *step changes* in the low-level (continuous) control schemes [92].
2. *The identification and separation of different forms of high-level control/reasoning.*
   Separate high-level control or decision making can capture a wide range of different reasoning aspects, most commonly ethics [15, 40] or safety [253]. Many of these high-level components give rise to governors/arbiters for assessing options or runtime verification schemes for dynamically monitoring whether the expectations are violated.
3. *The verification and validation of such architectures as the basis for autonomous systems analysis.*
   Fisher, Dennis, and Webster use the above structuring as the basis for the verification of autonomous systems [92]. By separating out low-level control and high-level decision making, diverse verification techniques can be used and integrated [84]. In particular, by capturing the high-level reasoning component as a *rational agent*, stronger formal verification in terms of not just 'what' and 'when' the system will do something but 'why' it chooses to do it can be carried out, hence addressing the core issue with autonomy [69].

Our reference three-layer autonomy framework consists of:

Reactions Layer — involving adaptive/reactive control/response aspects essentially comprised of low-level (feedback) interactions.

- **Challenges** At this level, the behaviour is driven by interaction with the environment, and the challenges to be addressed date back to the old debate between classic (symbolic) and behavioural (reactive) AI: in order for the system to work properly, it must be aware of the environment where it operates. A symbolic model of the environment requires facing the transduction problem, namely the accurate translation of the perceived portion of the environment into a symbolic representation. Transduction is often approximate and inefficient, and it is hard to exploit when the environment model requires frequent updates. To overcome the need for transduction, the environment may not be modelled at all, following a 'the world is its own best model' stance [44]. However, given that the environment must be known, in one way or another, to the system that operates therein, this approach requires hard-wiring the environment's implicit representation into the system's code, making the model difficult to inspect, reuse, and make scalable. Modelling the environment using some mathematical tool (equations in the continuous space, matrices, other more complex – but not symbolic – mathematical structures), may represent a viable trade-off between purely symbolic and purely reactive approaches.

- **Expectations** To properly address the challenges raised at this level, we would need operations to interact with (namely, perceive and act on) an environment whose interface is standardised, at least on a domain basis: this is where regulators would come into play. Different 'wrappers', or 'envelopes', for the implementation of these operations should be provided for different systems.
- **Examples** An autopilot is a good example of system operating at the Reactions Layer. It is "a device that keeps aircraft, spacecraft, and ships moving in a particular direction without human involvement" [48]. Hence, while being a conditionally autonomous system (not highly autonomous, as human pilots must be always ready to intervene, and not fully autonomous, as take-off and landing are operated by humans), it implements some complex *perception-action* rules, with no symbolic planning or reasoning in between.

Rules Layer — involving specific, symbolically-represented descriptions of required behaviours.

- **Challenges** The behaviours involved in such systems are tightly constrained by rules, that must be applied to a symbolic model of the environment and of the internal state of the system. The challenges raised by systems operating at this layer are those already mentioned for the Reactions Layer: who or what generates the model of the environment?

  Would the representation of the environment enable efficient validation with conditions of the real world? Would the reasoning and planning processes taking place over the model, be sufficiently efficient to produce results in a useful time frame? It is known that efficient general purpose planning with expressive actions is impossible [52] and that balancing reactivity and rationality is still an open problem [172]: despite more than 30 years of research on issues related to the Rules Layer, there is still a trade-off between the advantages of rule-based, symbolic approaches and their usability for real-time systems.
- **Expectations** Given that the transduction problem has to be solved on a system-by-system basis and no general tool or method can suit the needs of different domains, the scientific and industrial communities would at least benefit from having the rules for any domain of interest provided by the regulators in an explicit, logic-based form. As long as this need remains, the scientific community will face two transduction problems: the *environment-to-symbolic-model* on the one hand, and the *natural-language-regulations-to-logical-rules* on the other.
- **Examples** No existing industrial strength systems already operate at the Rules Layer, hence the example is fictitious: if the body of knowledge in the 'rules of the air' issued

by EASA or ICAO (the International Civil Aviation Organization) were represented in a formal, machine-readable way, autopilots could be extended with a rational component driven by such rules, and might reach a higher layer in our reference framework.

Principles Layer — involving high-level, abstract, sometimes philosophical, principles, often with priorities between them.

- **Challenges** At this layer, the specific behaviour is not prescribed but principles that can be applied to new/unexpected situations are provided. The main challenges are due to the fact that the environment where the system will operate is not fully anticipatable, and hence environmental modelling – be it implicit or explicit – is out of reach. Even if it were predictable up to some extent, the machine-readable representation of philosophical principles might require formalisms different from those the scientific research community is more familiar with.
- **Expectations** While regulators cannot provide formal rules governing this layer, they might at least suggest principles and priorities.
- **Examples** So far, the only systems exhibiting 'airmanship' are human pilots, who are not only able to fly an aircraft, but are also expected to manage ethical questions according to shared ethical (neither stated nor written, but 'known') principles. Pilots also need to cope with 'rational compliance to rules' issues at the Rules Layer, and with 'instinctive reaction to stimuli' issues at the Reactions Layer. These cross-layer connections are represented by vertical arrows in Fig. 6. An autonomous artificial system showing this kind of 'full airmanship', would be positioned at the Principles Layer in our reference framework.

  We believe that agents conceptualised in terms of mental concepts such as Belief–Desire–Intention [251] are the appropriate candidates for addressing principles and ethical issues that characterise this layer. The interactions among these mentalistic notions may lead to taking 'ethical' decisions on the basis of what the agent believes to be right ("saving the aircraft is right"; "saving human lives is right"; "saving human lives is more important than saving the aircraft"), what it desires to achieve ("saving both the aircraft and human lives"), and what it can do in practice ("if saving both the aircraft and human lives is not possible in the current situation, and saving human lives is more important than saving the aircraft, and by sacrificing the aircraft human lives will be spared, then the aircraft will be sacrificed").

We can split the high-level reasoning component further, into *rule-following* decisions and decisions based on *principles* (such as ethics). We distinguish these in that the former matches the required rules or regulations that the system should (normally) abide by while the latter is a layer comprising reasoning processes that are invoked when exceptional or unanticipated situations arise (and for which there are no prescribed regulations).

The key novelty here is the distinction between the normal operation where rules are followed (Rules Layer), and (unusual) situations where the autonomous agent needs to reason about whether to violate rules, using, e.g., ethical reasoning (Principles Layer).

3.2 What is needed from regulators

Currently, in most countries regulation responsibilities are distributed between legislation, providing the general framework within which an autonomous system is allowed to operate, and public authorities, which are responsible for providing detailed rules and to supervise the conformance of systems to these rules. In this paper, we focus on rule-making by regulatory agencies. That is, we do not discuss legal responsibilities of the designers, producers, owners, and operators of an autonomous system. We are concerned with behavioural aspects of such systems, and questions arising for regulatory bodies from their increasing autonomy.

In Section 2.3 we discussed the use of standards for verification, concluding that current approaches to certification and regulation are not adequate for verification of autonomous systems. In this section we briefly consider what would be needed from regulators in order to allow the standards to be used to verify autonomous systems.

A key issue is that current standards are not in a form that is amenable for formalisation and assessment of software, since they are oriented solely for use by humans. One way in which regulations are oriented towards humans, and do not readily support regulation of software, is that regulations are framed declaratively: a collection of statements that require substantial (human) interpretation. Another is that the regulations implicitly assume, and take for granted, human capabilities and attitudes. In order to certify autonomous software we need the scope of regulation to include not just low-level physical operation and sensors, but also higher-level decision-making. Finally, it would also be desirable for the plethora of relevant standards to be rationalised and consolidated. Consequently, it may be desirable to develop separate (new) standards for the assessment of software systems (e.g., software autopilots). At a high level, regulations should answer the following questions.

- **What does it mean for the system to be reliable/safe?** The answer to this question is a set of specifications, or the union of the following:
  - What are the regulations the system must obey? For example, the automated air traffic control system must always send a resolution to avoid two planes getting too close to each other whenever this is a possibility.
  - What emergent behaviours are expected? For example, the automated air traffic control system should keep the airspace free of conflicts.
  - What would be bad? For example: the assistive robot should never cause harm to a human; the Therac-25[6] should never deliver radiation to a patient when it was not activated by hospital staff; and the automated air traffic control system should never instruct two planes to collide with each other. These are often assumptions that can be hard to list. They are also *negative regulations*, i.e., complying with these is implicit in the regulations. We need to explicitly state them to enable robust verification efforts. Certification of autonomous systems goes in two directions: we need to know both that the system does what we want *and* that the system does not do what we do not want. This is particularly important for autonomous systems since the 'obvious' things that a human operator would know to never do tend to not be explicitly captured, but can be behaviours that a system should (or must) avoid.
- **How *busy* will the system be?** The answer to this question can be in the form of minimum/maximum throughputs, real-time bounds, or other measures. Essentially, the specifications need some environmental context. For example, the automated air traffic

[6] The Therac-25 was a computer-controlled radiation therapy machine, involved in at least six accidents between 1985 and 1987, where patients were given radiation doses hundreds of times greater than normal, resulting in death or serious injury [173].

> control system may vacuously assert that it can always keep the airspace free of conflict by grounding all aircraft, or limiting the number of flights. However, if the specification includes an indication of the minimum level of traffic that is expected (e.g., all flight take-off requests must be granted within a reasonable time bound modulo specific exceptions), then this can prevent the autonomous system from setting such inappropriate limits or learning undesirable behaviours. Such information, provided by regulators, might include bounds on how many aircraft need to be able to fly in the airspace, the maximum allowable wait time to be cleared to fly given safe environmental conditions, and other implied expectations.

Finally, specifications need to be compositional: they can be low-level and apply to one particular software routine or high-level and apply to the high-level architecture of the system. Because verification efforts are organised compositionally, as is safety cases coordination, there is a need to organise and divide the above list of types of specifications for each level/system component.

3.3 A process for identifying requirements for certification

We present a simple process that can be used to provide guidance in identifying properties that need to be specified as verification properties for certification. The key idea is that, if the autonomous system is performing tasks that are currently done by humans, then knowledge about how these humans are currently licenced can be used to help identify requirements. So, if the humans currently performing the task require some form of licensing (e.g., driver's licence, pilot's licence, medical licence, engineering certification, therapy certificate, etc), then carefully considering what the licensing process assesses and, then, how this might be assessed for an autonomous software system, would move a step towards their certification.

A motivating insight is that domains most likely to require (or benefit) from regulation and certification of autonomous agents are those domains where humans are very likely to have to be appropriately certified.

One challenge is that software and humans are very different in their abilities. Certain assumed characteristics of humans, such as common sense, or self-preservation, will need to be explicitly considered and assessed for software, even though they may not be assessed at all for humans. But even when a characteristic of the humans is assessed as part of a licensing regime, it may well need to be assessed in a different way for an autonomous software system. For example, a written exam to assess domain knowledge may work for humans, since limited memory requires the human to be able to reason about the domain to answer questions, but would not work for a software system that could merely memorise knowledge without being able to apply it.

We consider four key areas:

1. the licensing that is used for humans;
2. the assumed human capabilities (often unassessed) that are relevant;
3. the relevant laws and regulations, and what justifiable deviations might exist; and
4. the *interface* that artefacts (e.g., a cockpit) used by humans (and hence to be used by autonomous software systems replacing humans) presents.

We discuss these aspects in turn, beginning with licensing.

Licensing: We consider some of the qualities that licensing might assess, and for each indicate how we might assess this quality for an autonomous software system that is replacing a human:

- Physical capabilities (e.g., can execute the sequence of fine-tuned adjustments required to land a plane) – this can be assessed for autonomous software by simulation, and assessing specific component sub-skills.
- Domain knowledge (e.g., does the human know all of the protocols for safe operation, how to read and interpret domain-specific updates like Notices to Airmen (NOTAMs[7])) – for an autonomous software system, this would need to assess the ability to apply (operationalise) the domain knowledge in a range of scenarios designed to require this domain knowledge in order to behave appropriately. Note that this assessment could be in the form of a test (running some scenarios and observing the resulting behaviour), or in the form of formal verification (showing that certain properties always hold).
- Regulatory knowledge (e.g., does the human know all of the rules, such as restrictions on flying in different classes of airspace) – this can be tested similarly to domain knowledge.
- Ethical normalisation (e.g., does the human understand the importance assigned to the hierarchy of regulations from the regulatory body such as the FAA). An example would be that if an Unmanned Aerial System (UAS) is going to crash, the remote human operator needs to understand that it is better to clearly communicate to the appropriate air traffic control authority that the UAS will violate a geofence surrounding an unpopulated area and proceed to do that. The alternative, obeying the geofence but crashing the UAS into a populated area, is not acceptable – for autonomous software, one could verify certain priorities, if the reasoning is explicitly represented in a decision-making component, or assess performance in scenarios designed to present these sort of ethical challenges.

Assumed human capabilities: Various characteristics of humans may not be assessed at all, but simply assumed, since they are sufficiently universal, and, for some characteristics, it is very clear if they are absent; e.g., a human lacking physical mobility (lacking requirements for *physical capabilities*), or being a child (lacking requirements for advanced *ethical normalization*) would be clear without requiring explicit assessment. Specifically, in considering assessment of autonomous software systems, we would want to carefully consider what human characteristics are required and assumed to hold, without any assessment. Typical questions might include:

- Does the human need a certain education or level of understanding of, e.g., mathematical concepts, or medical or common-sense knowledge about characteristics and capabilities of humans that is obtained by being human rather than by certification? For example, do we assume a pilot knows that, when flying a passenger plane, the passengers require a certain amount of time to fasten seat belts.

[7] A NOTAM is "a notice distributed by means of telecommunication containing information concerning the establishment, condition or change in any aeronautical facility, service, procedure, or hazard, the timely knowledge of which is essential to personnel concerned with flight operations." [127]

- Does the human need assumed (but untested) physical capabilities? For example, a pilot can sense when something is wrong from certain sounds or vibrations in a plane that may not be present in simulations or ground tests for certification.
- Does the human need a certain level of maturity or life experience? For example, a sound may be readily identifiable as 'bad' even if the pilot has never heard it previously.
- Do we assume basic properties of human values/ethics, such as that the pilot would never crash the plane on purpose because the pilot has a strong inclination toward self-preservation? Do we assume an operator would never choose to hurt others?

On the other hand, certain human characteristics that need to be assessed when certifying humans may not need to be considered when certifying software. For instance, psychological state, or personality aspects (such as being aggressive, having a big ego and therefore being over-sensitive to criticism of flying ability, or being impulsive) should not need to be considered for autonomous agents.

Legal factors: Often licensing includes testing for knowledge of relevant laws and regulations. We consider legal factors separately because this is a key source of specifications, and because the licensing assessment may not cover all relevant regulations and laws.

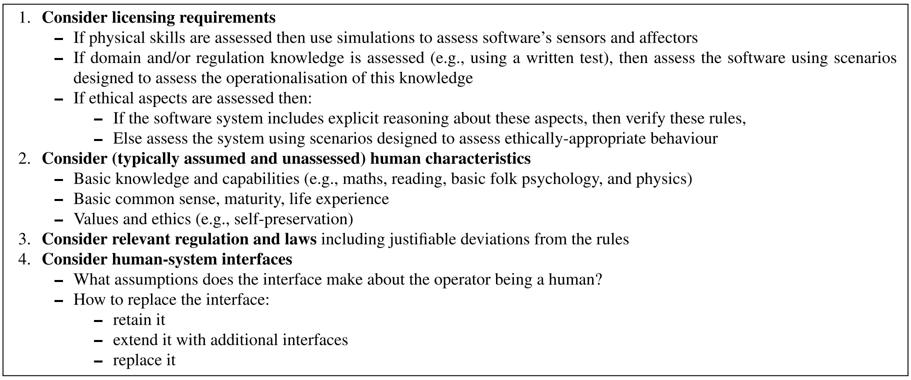
As per the three layer framework, we need to identify not just the 'by the book' rules (e.g., regulations and laws). Rather, we also need to consider situations where these rules may need to be (justifiably) overridden, and the principles that would drive such decisions. The key questions are: in what situations should the rules be overridden? How can the system identify these situations? How can the system decide what to actually do in these situations? More specific questions to be considered include:

- What situations could lead to outcomes considered to be 'bad', 'unsafe', 'insecure', or otherwise universally to-be-avoided if at all possible? How bad are each of these? Are some worse than others? If there is a choice, is there any ranking or cost (possibly probabilistic) that can be associated with each? For example, if an autonomously-operating UAS is going to crash and must choose between crashing into a pile of trash on a curb or the car parked next to the pile of trash, the cost function would be expected to steer the crash landing toward the pile of trash. This could be defined in terms of minimising the repair cost of the affected property. One might furthermore define the cost of harming a human in a crash as higher than any situation where a human is not harmed.
- Are some rules more important to obey than others? Are there divisions of the rules into hard constraints versus soft constraints? Is there some partial ordering on the importance of the rules?
- Are there any acceptable reasons to break the rules?

In order to develop a system that can meet the requirements, we need to also consider what are the computational requirements of the system. What does it need to be able to measure, deduce, or decide?

Note that context is often left unspecified but it importantly restricts the applicability of licensing. To take context into account, we identify the licensing requirements, and then, for each requirement, consider whether it is relevant for the system. For example, for an auto-pilot that is only used while the plane is at cruising altitude, we would not need to consider requirements related to landing, or interaction with ground obstacles, even though these are requied for a human to earn a pilot's license.

Human-system interface: There is a collection of similar factors that relate to the interface that a human currently uses to interact with artefacts. Such artefacts are designed for



1. **Consider licensing requirements**
   - If physical skills are assessed then use simulations to assess software's sensors and affectors
   - If domain and/or regulation knowledge is assessed (e.g., using a written test), then assess the software using scenarios designed to assess the operationalisation of this knowledge
   - If ethical aspects are assessed then:
     - If the software system includes explicit reasoning about these aspects, then verify these rules,
     - Else assess the system using scenarios designed to assess ethically-appropriate behaviour
2. **Consider (typically assumed and unassessed) human characteristics**
   - Basic knowledge and capabilities (e.g., maths, reading, basic folk psychology, and physics)
   - Basic common sense, maturity, life experience
   - Values and ethics (e.g., self-preservation)
3. **Consider relevant regulation and laws** including justifiable deviations from the rules
4. **Consider human-system interfaces**
   - What assumptions does the interface make about the operator being a human?
   - How to replace the interface:
     - retain it
     - extend it with additional interfaces
     - replace it

Fig. 7 Summary of the process for identifying requirements

human use, and if a human performing the task is going to be replaced by an autonomous software system, then whether the existing interface presented by the artefact embodies assumptions about humans should be taken into account. Specifically:

- Does the interface assume physical shapes of humans, such as being operated by a human hand?
- Does it assume physical limitations of humans, such as having a minimum reaction time or a maximum speed for selecting multiple controls in sequence?
- Does it assume mental limitations of humans, such as taking into account that a human cannot instantly take control of a system but requires orientation to the operational context to make reasonable decisions?
- Does it assume that human-type faults will occur, such as being designed to avoid human confusion modes?
- Does it assume that common sense deductions on the part of the operator are automatic? For example, a human pilot will automatically notice if the wing detaches from the aircraft and there is no explicit requirement that the aircraft operator must continuously ensure the aircraft has both wings but an autonomous system would need to be explicitly designed to consider this a fault (not just the instability it causes) and, e.g., avoid future control decisions that would only work if the aircraft had two wings [238]. There is not a sensor for every 'obvious' fault yet an autonomous system needs to account for all faults that are obvious to humans, even when they trigger fault warnings for unrelated errors.[8]

There are three options to deal with a situation where a human is being replaced (partially or completely) with software, and the human interacts with an existing artefact using an interface. These options are: to retain the interface, and have the software interact with it; to extend the artefact with additional interfaces for the software; or to replace the artefact's interface completely. The process for identifying requirements is summarised in Fig. 7.

[8] This is not an artificial example: there was a case of an engine exploding, resulting in a large number of seemingly unrelated alerts [82]. Thankfully, in that instance, the human pilot realised what had happened, and was then able to safely land the plane.

3.4 Verification of autonomous software systems

In Section 2.2 we observed that the most solid and reliable way to certify autonomous software systems would be the adoption of *formal methods*.

Formal methods are mathematically rigorous techniques for design, specification, validation, and verification of a wide variety of systems. They contribute to certification in a number of ways, from requirements design, to checking that all requirements can be true of the same system at the same time (before the system is built) [216], to verifying that early (or even partial) designs always satisfy requirements, to generating test cases, to checking during system runtime that the system is still meeting requirements.

In order to verify that a system operates correctly, we need to know how the system works. If we do not have sufficient information on both how the system operates and what it means for the system to operate safely, then it is impossible to verify that these two behaviour spaces match up with each other. Knowing how the system works includes knowing sufficient implementation details to be able to certify correctness.

Though each method works differently, intuitively all formal methods verify that the system – or a model of the system – does what we expect it to do and nothing else. This capability is required for certification, e.g., via *safety cases* per the standards of the aerospace industry, where certification requires making a case, backed by evidence, that the system meets standards for safety and reliability.

In some formal methods, such as model checking and theorem proving, both the system under certification and the properties to be verified are modelled using a rigorous, mathematical or logical language. We indicate these methods as *formal at the property and system level*. The system model itself, however, is necessarily an abstraction of the real system and hence it is incomplete: any results from methods that operate, albeit in a rigorous way, on abstractions of real systems, should be validated against the actual environment. (Note that many modern model checkers and theorem provers are now capable of generating software from their proven models that can be used as a basis for, or as the entire, real system implementation thus enabling straightforward validation.) Other methods model the properties to check using rigorous formalisms and languages, and check the property against the *real* system under certification. We define these methods, which include some static analysis approaches and runtime verification, as *formal at the property level*. *Semi-formal methods* specify either the system or the properties that they should meet using languages with informal or semi-formal semantics. UML [33] and Agent-UML [21] are examples of semi-formal specification languages, as well as test cases. *Informal methods* are based on specifications provided in natural language and are out of the scope of our investigation.

In this section we review five verification methods: model checking, theorem proving, static analysis, runtime verification, and systematic testing. The first four of these are usually categorised as formal or semi-formal methods. Software testing is not a formal method, but it is one of the most widely adopted verification approaches in industry, often associated with quality assurance. The 'Software Testing Services Market by Product, End-users, and Geography – Global Forecast and Analysis 2019–2023' [111] foresees that the software testing services market will grow at a compounded average growth rate of over 12% during the period 2019–2023. Software testing – once automated and seamlessly integrated in a development method, as in the 'continuous integration model' – may indeed represent a useful complement to formal methods. Testing can be measurably improved by using artefacts from formal verification for test-case generation. However, as is well-known, testing is not exhaustive, which limits its utility.

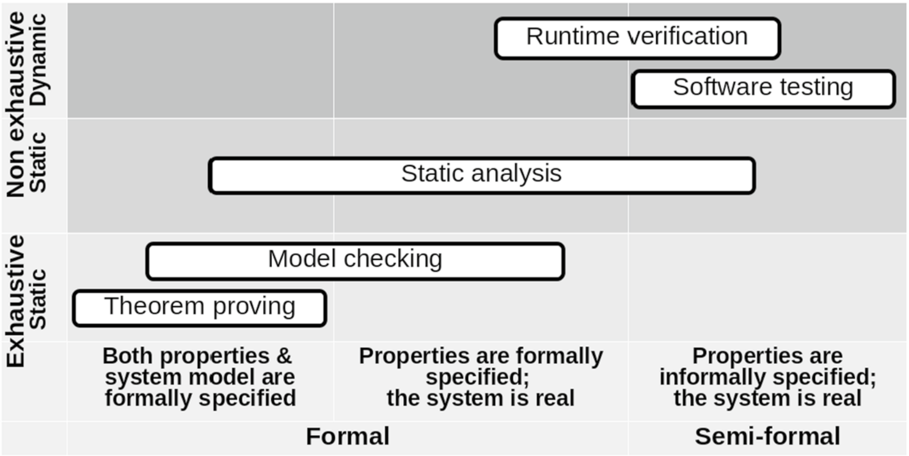

Non exhaustive
Dynamic
Static
Exhaustive
Static
Runtime verification
Software testing
Static analysis
Model checking
Theorem proving
Both properties & system model are formally specified
Properties are formally specified; the system is real
Properties are informally specified; the system is real
Formal
Semi-formal

Fig. 8 Knowledge map of the verification methods discussed in Sect. 3.4

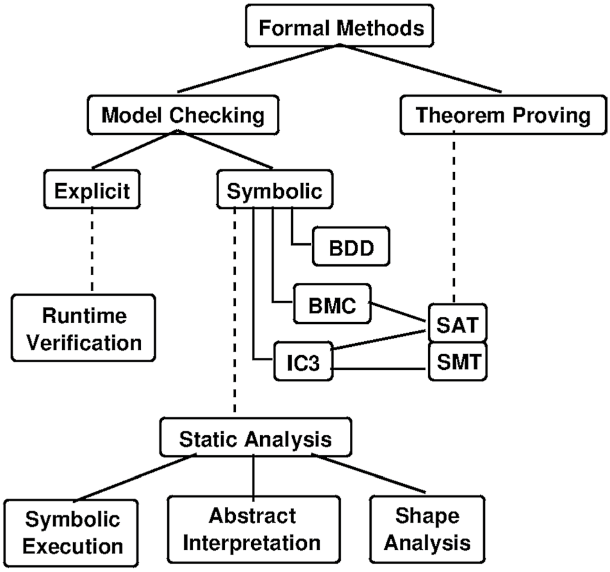


Fig. 9 The tree of formal methods; solid lines represent direct variations whereas dashed lines represent related derivations. Symbolic model checking [217] is divided into three major algorithms under the hood: BDD (binary decision diagram), BMC (bounded model checking), and IC3 (a pseudo-acronym sometimes replaced with PDR for property-directed reachability). As a subtlety, the BMC and IC3 algorithms are not complete by themselves but many tools use these acronyms to refer to those algorithms supplemented with proofs to make them complete. Both model checking and theorem proving commonly use SAT (Boolean satisfiability) and SMT (satisfiability modulo theories) solvers under the hood

Figure 8 positions the five methods in a space characterised by the *formality*, *exhaustivennes*, and *static vs. dynamic* dimensions. Figure 9 details the formal methods and their relationships.

A method is *exhaustive* when it verifies the entire behaviour space, over all possible inputs, including for an infinite input space. Exhaustive verification includes proving both the existence of ‘good’ or required behaviours and the absence of ‘bad’ or requirements-violating behaviours, something that can never be demonstrated through testing or simulation. Being exhaustive refers to the capability of exhaustively exploring the state space generated by the system model. It is not currently possible, except for a small class of systems with specific characteristics, to be exhaustive with respect to a system of realistic industrial size, unless we employ compositional verification. Compositional verification involves decomposing a verification problem into sub-problems with well-defined relationships such that we can prove that the verification of the set of sub-problems implies the verification of the system as a whole [57].

A method is *static* when it is applied on the system's code, or on its abstract/simplified model, without needing the real system to be executed. A method is *dynamic* when it operates while the system runs.

Model Checking: This method performs an exhaustive check that a system, given in a logical language, always satisfies its requirements. Intuitively, model checking explores all possible executions of the system, checking the system model cannot reach states that violate the provided verification properties.

- **Inputs** (1) a logical description of the system (either a model in a logical language such as SMV [185] or AIGER [30], or the system itself, e.g., for software systems); (2) a requirement in a temporal logic such as LTL. See [217] for a survey with examples.
- **Outputs** (1) an automated (push-button) proof (usually by assertion, without the full proof argument) that the system always satisfies the requirement; or (2) a counterexample, which is an execution trace of the system from system initialisation that shows a step-by-step path to a system state where the system has operated faithfully to its definition yet still violated the requirement.
- **Strengths**
  - **Automated.** Counterexamples are excellent maps for debugging and are generated without guidance from the user.
  - **Exhaustive.** Verification reasons over the entire state space.
  - **Verification of Absence of Bad Behaviours.** Model checking both gives the existence proof that a system always does what we expect given the inputs *and* that the system never does something we don't expect for any input. In other words, given an appropriate specification, it verifies the absence of behaviours that we don't want, whether they are triggered by an input or no input at all.
  - **Incremental.** Model checking can efficiently analyse partial models/partial systems; can be used from the earliest stages of the design process to save time pursuing designs that cannot satisfy their requirements based on the partial design-so-far.
- **Weaknesses** Model checking is garbage-in, garbage-out; the results are only as accurate as the input system/model and the input requirement. One approach for validating the results of model checking against the actual environment by recognising assumption violations at runtime has been proposed in [89, 90], but the example addressed there is simpler than any real autonomous system. Modelling a system in a logical language and specifying a requirement in a temporal logic are the biggest bottlenecks to the use of model checking [218]; both tasks are difficult to complete and to validate. It is easy to create models that are too large to analyse with current technology, such as large pieces of software or models over large numbers of variables. Not all logical systems can be specified in languages that can be model-checked.
- **Applicability** model checking is widely used for both hardware and software, such as that seen in the Rules Layer[17, 55, 56, 109]. System protocols are particularly amenable to model checking; it has successfully verified real-life, full-scale communication protocols [75, 190], air traffic control systems [98, 183, 262], wheel braking systems [37], and more. Model checking has been used for the Reactions Layer, but quickly

reduces to either hybrid model checking or the verification of coarse abstractions [2, 5, 106, 162]. Hybrid model checking can involve a wide array of numerical techniques aimed at solving, for example, differential equations concerning control or environmental models. For the Principles Layer, model checking has been used for verifying BDI agents [34–36]; epistemic and temporal properties of multiagent systems [170, 171, 195]; the agents' knowledge, strategies, and games [177]; and general, high-level decision making [69] that has been extended to ethical principles [40, 70].

We note one important variation, *program model checking* [244]. Per its name, program model checking replaces the abstract model with the actual code/software system. It uses the same model-checking algorithms, but on symbolic executions of the system. The advantage of such an approach is that it simplifies model validation (e.g., proving the link between the model and the real program/system) since we directly check the real program. However, a disadvantage is that the use of techniques such as *symbolic execution* can make the checking process slow and complex. Finally, note that the Principles Layer model-checking of high-level, and ethical, decision-making described above [40, 69, 70] actually utilise program model checking over rational agents [66, 68].

Theorem Proving. A user creates a proof that a system satisfies its requirements that is checked by a known-to-be-correct proof assistant. Some automation of standard logical deductions is included in the proof assistant.

- **Inputs** A theorem asserting a requirement holds over the system; the same large proof includes relevant definitions of the system and (sub-) requirements, and any necessary axioms.
- **Outputs** A computer-checked proof that the theorem holds. If no such proof can exist (e.g., because the theorem does not hold), the user might gain the necessary insight into why this is the case from the way in which the proof fails.
- **Strengths** Theorem proving can provide strong type-checking and generate proof obligations that provide unique insight into system operation. The proofs generated are replayable: they can be re-run by any future user at any time. If the input theorem changes (e.g., to perform a proof over the next version of the system), the old proof can be replayed (in some contexts) and only prompt the user for input where small changes are needed to update the proof to the new theorem. Proofs can be built up in extensible libraries and hierarchically inherit from/build upon each other. Large swaths of mathematics important to assertions over safety-critical systems have been proved in free libraries associated with the most popular theorem proving tools. Theorem proving can, and has, checked very large systems, even beyond what humans can inspect, such as the four-colour theorem [12].
- **Weaknesses** This formal method requires the most experience from users and has a comparatively high learning curve relative to other forms of verification. Setting up a proof correctly is a difficult activity even for trained experts. The user is the sole source of insight into why proofs fail; there is no automated counterexample like with model checking.
- **Applicability** Theorem-proving has been widely used for software seen in the Rules Layer, and techniques such as Hoare Logic [107] and tools such as Isabelle [194] are often used there. Theorem-proving has been used for the Reactions Layer, most notably verification of NASA's aerospace systems via PVS [97, 188, 189, 199], verification of, e.g., automotive protocols through the KeyMaera system [201, 202], and the use of theorem-proving tools to prove control system stability [159]. See [45] for

an introduction to PVS specification for a Boeing 737 autopilot. Theorem-proving for the Principles Layer is less common, though it has been tackled [41].

Static Analysis. During system design time, an automated tool examines a program or code fragment without executing it, to check for common bugs. Static analysis can be combined with techniques for expanding the types of bugs found, including symbolic execution (analysing a program's control flow using symbolic values for variables to create a map of program execution with provable constraints), abstract interpretation (creating an abstraction of the program's semantics that enables proofs about its execution), and shape analysis (mapping the program's memory signature through its data structures).

- **Inputs** (1) Program or code fragment to analyse; (2) requirements to check (similar to model checking) or a selection of common code errors, usually provided by the chosen static analysis tool.
- **Outputs:** List or visualisation of possible code defects.
- **Strengths:** Automated tools analyse the structure of the code rather than just paths through the code like testing; they can therefore provide a deeper analysis with better code coverage. Static analysis is faster and more reliable than the alternative of manual code review. Though performance depends on the programming language and static analysis tool, performing static analysis is usually totally automated with easy-to-understand output suitable for a non-expert in verification.
- **Weaknesses** Static analysers have no understanding of developer intent (e.g., they can detect that a function may access unallocated memory but will not find that the function does a different task than intended/labeled in the comments). They cannot enforce full coding standards, e.g., because some coding rules are open to interpretation, like those concerning comments, good documentation, and understandable variable names. The above ambiguities (in programmer intent, coding standards, use cases, etc) can lead to false positive and false negative defect reports. False negatives mean that the analyser has failed to identify an error, and can lead to fallacious trust in the software. False positives mean that the analyser reports an error where there is none, and can lead to wasted work on the developer's side.
- **Applicability** Static analysis covers a range of different methods. Basic checks, i.e., whether each used variable was previously initialised, are included in most modern compilers. More sophisticated checks, e.g., whether pointer access may lead to an exception, are realised by widely used tools such as Lint [95]. High-end static analysers such as Polyspace [182] and Astrée [59] are able to check for general assertions. A review of 40 static code analysis tools updated to November 2019 is available via the Software Testing Help blog [236]. According to CENELEC EN50128, static analysis is highly recommended for all safety-relevant software. Consequently, static analysis has been applied for most safety-critical software in avionics, railway, and automotive; e.g., NASA regularly releases its own home-grown static analysis tools including (Java) Symbolic PathFinder [179], IKOS (Inference Kernel for Open Static Analyzers) [39], and C Global Surveyor (CGS) [38]. Static analysis techniques generally analyze software in the Rules Layer. The Reactions Layer requires analysis techniques able to deal with continuous physical models, and the Principles Layer requires the ability to deal with potentially complex reasoning. Although static analysis can analyze autonomous systems' software, its key weakness, being non-exhaustive, limits the utility of applying it.

Runtime Verification. Runtime verification (RV) is a semi-formal method aimed at checking *the current run* of the system. RV is essentially model checking, but instead of checking that the system model always satisfies the given behaviour requirement, RV checks only that given system run(s) satisfy this requirement. RV is most often run online (e.g., embedded on-board the system during operation), and stream-based (e.g., evaluating requirements continuously throughout system operation), though it can also be run offline (e.g., with recorded data for post-mission analysis).

- **Inputs** (1) Input data stream or streams containing time-stamped sensor or software values, e.g., sent over a system bus to the RV engine; (2) A requirement to verify, expressed in the form of a temporal logic formula, most commonly Mission-time Linear Temporal Logic (MLTL) [175, 176, 213] or First-Order Linear Temporal Logic (FOLTL) [20, 76] whose variables are set by the input data or operationally via finite state automata, trace expressions [8], etc (see Sect. 2.1 of [83]).
- **Outputs** For online, stream-based RV, the output is a stream of $\langle \text{time}, \text{verdict} \rangle$ tuples containing a time stamp and the verdict from the valuation of the temporal logic requirement (true or false), evaluated over the data stream starting from that time step. Variations on RV include extensions to non-Boolean (e.g., Bayesian or otherwise probabilistic) results, and evaluations of only part of the data stream of interest.
- **Strengths** RV is the only formal verification technique that can run during system deployment. Sanity checks can provide near-instant confirmation that a system is obeying its requirements or that a failure has occurred, providing an efficient mitigation trigger. Like simulation, RV can also provide useful characterisations of system runs, though RV analyzes runs in terms of violations of formal specifications; see [219] for a detailed comparison.
- **Weaknesses** Specification of the requirement to monitor at runtime is the biggest bottleneck to successful deployment of RV [218]. Specifications are difficult to write for the same reasons as for model checking; further complications include noisiness of sensor data and challenges of real-world environments. Limitations and constraints of embedded systems and certification processes challenge RV implementations, though to date three tools have risen to these real-world-deployment challenges and more are sure to follow [3, 196, 220]. Of course, violation of certain critial safety properties is not acceptable, even if this is detected. Since RV operates at (or after) runtime; it is not suitable for such properties.
- **Applicability** RV is widely used for software in Rules Layer [102, 215, 233]. RV of multiagent systems using trace expressions [9, 88] is also a viable approach to cope with the Rules Layer. RV has been used for the Reactions Layer, for example in [19]. The R2U2 framework [99, 187, 213, 220, 226] is a real-time, Realisable, Responsive, Unobtrusive Unit for runtime system analysis, including security threat detection, implemented in either FPGA hardware or software. R2U2 analyzes rules and reactions spanning the Rules Layer and (with the addition of a parallel system model) the Reactions Layer of our reference framework. For the Principles Layer, [63] surveys the use of norms for monitoring of the behaviour of autonomous agents: monitoring of norms is indeed foundational for processes of accountability, enforcement, regulation, and sanctioning. The idea of a runtime ethical monitor, or governor, is also commonly used [15, 253].

Software Testing. Software Testing (ST) is not a formal method, though formal methods are often used for test-case generation with provable qualities, such as coverage metrics. ST amounts to observing the execution of a software system to validate whether it behaves as intended and identify potential malfunctions [29]. Testing can take place at different levels including Unit Testing, which tests individual units of source code; Integration Testing, aimed at testing a collection of software modules seen as a group; System Testing, carried out on a complete integrated system to evaluate its compliance with respect to its functional and/or system requirements; and User Acceptance Testing, meant to determine if the system meets its requirements in terms of a specification or contract, and allow the user to accept the system, or not.

- **Inputs** Depending on the test level and on the testing technique adopted for the level, the inputs that feed the ST application can change dramatically. For example, Unit Testing tests source code, while System Testing may follow a black-box approach, with the stucture of the system unknown to the tester. All testing methods need some requirement to test, either explicitly stated or implied; this can be the same requirement utilized in model checking, static analysis, or runtime verification.
- **Outputs** As for RV, ST always consists of observing a sample of executions, and giving a verdict over them. The output may be an association between a test case, an execution run, and the respective verdict.
- **Strengths** ST can be performed in an agile way [23] and can be directly integrated in the software development method [22], allowing the software to be developed and tested in the same development cycle. ST techniques may be very efficient; coverage analysis and reliability measures can help in performing 'as complete as possible' testing.
- **Weaknesses** Software Testing is not exhaustive. Identifying the most relevant units, functions, or functionalities to test is not easy, and finding the right balance between testing enough, but not too much, requires a deep understanding of the system under test. In particular, testing autonomous systems is hard since the range of possible behaviours that can be exhibited by an autonomous agent can be vast [250, 251]. The design and development of test cases is expensive; if the system under test changes, test cases developed for a previous version might not be reusable for the new one. Importantly, ST can never verify the absence of bad or undesirable behaviours, only the (partial) existance of desirable ones. Also unlike formal methods, ST requires a complete, executable system to test, pushing ST verification later in the design life cycle than formal verification, which easily analyzes partial, early system designs.
- **Applicability** While software testing is commonly used in industry for both the Reactions Layer and Rules Layer, and is applied for testing features at both levels in autonomous [105, 227] and multiagent systems [192], we are not aware of proposals for testing the Principles Layer of an autonomous system.

The analysis that we carried out shows that no silver bullet exists for addressing the certification, via formal and semi-formal verification, of autonomous systems. All the methods considered in this section show strengths and weaknesses. Also, the applicability to the Principles Layer is not widespread yet.

Table 7 Examples of autonomous systems

System	Scope of Autonomy	Targeted future autonomy	Complexity of decision making	Potential harm	Amount of existing regulation
Robot vacuum cleaner	low	high	low	none	low
Autonomous trading system	low-medium	high	low	high	low-medium
Driverless train	low-medium	full	medium	very high	high
Unmanned aerial system	medium	full	high	very high	medium
Self-driving car	medium-high	full	high	high	low-medium
Personal assistant agent	high	full	very high	low-medium	low
Home assistant robot	high	high	very high	medium	low

4 Illustrative examples

In this section we discuss some examples demonstrating the generality of the framework. The authors are aware that currently these systems are not likely to be certified; however, this might change in the future.

We exemplify systems that can operate independently, and argue that we can extend the framework for the case of operation in a group. For example, we can consider a single autonomous aeroplane, or a swarm of UAS; drive a car, or do car platooning; interact with a stand-alone personal assistant agent (PAA), or use this as an interface to a complete smart-home environment.

Table 7 contrasts over several dimensions some examples of autonomous systems in various domains: the same systems/domains were illustrated in Fig. 5. Note that the second column of Table 7 indicates the (current or near future) *scope* of autonomy (based on Fig. 5), whereas the next column indicates the (somewhat longer-term future) *level* of autonomy.

We remind the reader of the levels of autonomy introduced in Sect. 1.1, Definition 1: no autonomy; low autonomy; assistance systems; partial autonomy; conditional autonomy; high autonomy; full autonomy.

Our examples are distinct and, in particular, differ with respect to the complexity of their decision making. By this, we mean the amount of information used to determine the system's behaviour in any given situation. Usually, this complexity is known only to the designer of the autonomous system, it can not be observed from the outside. The most straightforward decisions are based directly on inputs of certain sensors such as ultrasonic distance sensors, lidars, cameras, etc. A typical decision would be "if there is an obstacle to the left, then move to the right." We consider such low-level decisions, where the action is derived directly from the sensory input, to be of low complexity. Additionally, however, the system can have some built-in knowledge concerning its environment that influences its behaviour; this could be map data, time tables, situation databases, rule books, etc. The internal state of an autonomous agent can be modelled in terms of its beliefs, intentions and desires. We consider algorithms that rely on extensive models of the environment to be of medium or high complexity. More sophisticated algorithms take the history of decisions into account, and thus 'learn from experience'. Such algorithms can adapt to changing environments and tasks, evolving constantly. Thus, even the designer of a system can not predict its behaviour with certainty if the system has been operating for some time. We consider such decision making to be of very high complexity.

The examples differ also with respect to their potential harm, here defined generally as 'negative outcomes' to individuals or society. One aspect of this is safety — the absence of physical harm. The international standard IEC 61508 on functional safety defines five safety integrity levels (SILs). SIL0 means there are no safety related requirements, SIL1–4 refer to low, medium, high, and very high safety-related requirements, respectively. Other domains such as aerospace and automotive have come up with similar safety classifications. The categorisation of a (sub-)system into a certain SIL class is done by considering the risk that the system imposes on its environment. This is the likelihood of occurrence of a malfunction multiplied by its possible consequences.

Systems/scenarios with higher levels of potential harm will require strong regulation. So, both in Table 7 and later subsections, we have highlighted the level of regulation available. In most cases, this regulation does not mention 'autonomy' nor consider that the human might not be 'in control'. We would expect that systems with high potential risk will have stronger forms of regulation. The *current* amount of available regulation for each of the sample systems is indicated in Table 7, ranging from 'low' (scattered or unsystematic regulations; points of inconsistency or obsolescence in regulations) to 'high' (comprehensive regulatory framework; consistent and current regulations). In addition, once autonomy is introduced, and especially where this can impact upon safety or other potential harms, then enhanced regulations for this aspect will be essential.

In the following, for each example we first describe the functionalities, and then position the functionalities of the example with respect to the three layers in our reference autonomy framework. We then describe the trends in the domain, in particular around the future level of autonomy per Definition 1. Subsequently, we comment on the safety-criticality of the example, and hence the amount of likely needed and available regulation. Finally, we comment on suitable verification, validation, and analysis techniques for the example.

4.1 Simple case: Robot vacuum cleaner

Functionalities We begin with a simple example: a domestic robot tasked with simple objectives, such as vacuuming or sweeping the floor, or cutting grass in the lawn.

Positioning with respect to the layers in our reference autonomy framework In terms of the three layer model, this sort of domestic robot illustrates that in some situations, not all of the layers are needed. There is, of course, a need for a Reactions Layer, which deals with physical aspects of the environment (sensing, positioning, locomotion, and physical actions). There is also a need for a Rules Layer that follows well-defined rules for 'normal' situations. Such rules might specify a regular schedule for cleaning, as well as indicating when this schedule might be abandoned or changed (e.g., due to human request, or the system detecting a situation in which cleaning ought to be postponed, such as a dinner party). The vacuum cleaner must limit its maximum speed to avoid harm, change direction if it encounters an obstacle, stop if it detects a fault of any kind, and obey the boundaries of its electric fence. The Principles Layer of a vacuum cleaner is trivial, since the system's functionality is relatively simple. Specifically, we do not expect the system to deal with any unusual situations that would require it to override the 'normal situation' rules. The cases when the robot is stuck, or jammed, or too hot due to unexpected warming, are 'normal situations', addressed by the simple 'switch off' rule. The only way the application of this rule could harm, would be to cause a human to stumble into the robot, but in this scenario we are assuming that humans can cope with the presence of the robot in their home, which means, they have the ability to perceive and avoid it.

Level of (future) autonomy According to our categorisation, such robots are designed to operate autonomously in their respective environments. Human assistance is necessary only for the initial set up, for emptying the dustbin, refilling cleansing fluids, maintaining brushes or blades, and repositioning the robot in case it is stuck or out of battery power. However, the complexity of decision making in these devices is low: often, the control programs are based on a few sensory inputs only. Many of these robots navigate the area in a random fashion, turning upon contact with another object, and moving towards the docking station guided by an infrared beam if the power is low. More sophisticated models can learn a map of their operating environment and use advanced path planning methods to traverse the whole area. There are no complex interactions with other devices or actors in the environment. The scope of autonomy is also low, given the limited autonomous functionality of these systems.

Safety criticality (of autonomous aspects) For this example we exclude the possibility of the tasks having any direct safety consequence (such as caring for ill humans, having responsibility for monitoring for injury, or providing medication), and we assume that the humans interacting with the robot are aware of its existence and can cope with it in a ‘normal’ way. Therefore, the safety criticality of the devices is low.

Amount of available regulation (for autonomous aspects) With respect to available regulation, there are just the general laws governing the use of electric equipment, batteries, etc.

Suitable verification, validation, and analysis techniques Cleaning robots are often adopted as simple and affordable examples to show the features and potential of different programming languages, tools, and even verification and validation techniques [61]. Recently, four of the latest premium robot vacuum cleaners were analyzed and compared by IoT experts according to security and privacy issues [16], but no formal techniques were used there. Concerns have arisen since some companies wanted to use data collected by such home devices — layout of the house, operating hours etc — for marketing purposes. While we agree that all the techniques reviewed in Sect. 3.4 are suitable for checking that the robot Reactions Layer and Rules Layer behave as expected, their application on real systems is often considered not to be worth the effort, given the absence of safety issues. However, these devices make good exercises for academic education in these techniques.

4.2 Autonomous trading system

Functionalities Autonomous trading systems (ATSs) operate in financial markets, submitting buy and sell orders according to some trading strategy. ATSs have a history back to 1949, but came to widespread attention after the ‘flash crash’ on the US stock exchange in 2010. By 2014 it was reported that in excess of 75% of trades in public US security markets were by ATSs [174]. An ATS submits market instructions, primarily in the form of a buy or sell order. It may decide when to do so, and the commodity, quantity, and bid or ask price. ATSs can operate in any market where they are permitted, such as a stock market or an energy market. In notable contrast to un-aided human traders, ATSs can operate high-frequency trading. The price prediction techniques and trading strategies vary. For example, the stock price of a single company might be predicted using a non-linear function approximator [18], and the trading strategy might be based on a set of rules.

Positioning with respect to the layers in our reference autonomy framework With respect to the layers in our reference autonomy framework, we can locate the communications protocols and simple but fast responses to market stimuli ("if the price is below threshold, then issue a sales request") at the Reactions Layer. More sophisticated trading rules and strategies, which take general market evolution and developments into account ("if there is a shortage of oil, then coal prices are likely to rise"), are located at the Rules Layer. Although we might anticipate a possible Principles Layer that reasoned about unusual situations, such as suspending the usual trading strategy in the face of a stock market crash, the speed of trading makes this unlikely, and in practice, we would not expect a Principles Layer to be used.

Level of (future) autonomy Current ATSs are already capable of operating autonomously, without moment-by-moment human intervention. Nonetheless, ATSs can be used also as an assistant by a human trader, for example to automatically build or maintain a position specified by the trader. It is likely that the degree of autonomous operation and the time-scale of operation will increase, whereas humans become less able to oversee the trading behaviour except at a high level. The scope of autonomy depends on the sophistication of the ATS, and varies from low (for relatively simple ATSs that follow simple trading rules), to medium for more sophisticated ATSs.

Safety criticality (of autonomous aspects): Given the disruption to financial markets from ATSs, they have a high level of potential harm, and hence are considered safety-critical. Issues of concern with respect to safety include software errors, herd trading, market fragility, manipulative market strategies, lack of scrutability, the need for an algorithmic 'kill switch', and fair market access.

Amount of available regulation (for autonomous aspects): Given the safety criticality of ATSs, the needed regulation is high. Regulation has increased since ATSs became a standard tool in the financial sector in the 21st century. In the US, the financial authorities introduced stricter rules following the 2010 crash. These pertain to algorithmic trading behaviour, but also to the use of ATSs more generally, and the responsibility of the companies using ATSs to have adequate supervision of ATSs [91].

Suitable verification, validation, and analysis techniques

These ATSs are essentially feedback control systems. As such, standard analysis techniques from control systems such as analytic stability proofs or algorithmic hybrid systems verification [106, 202] could be deployed. However, such approaches require a very good model of the environment, and its reactions, in which the trading system resides. In particular, the environment includes other ATSs, and due to commercial sensitivity, little is likely to be known about them. Without a precise environmental formalism, verification and validation techniques are of little value. Testing can be applied in this area but, again, strong verification cannot be achieved. Consequently, there is a danger of 'run-away' feedback loops that are impossible to predict beforehand.

4.3 Driverless trains

Functionalities: Automatic railway systems for the transportation of people have been in use since the beginning of the 21^{st} century. Generally, these systems operate on dedicated routes, which are protected from human access by special barriers. Moreover, often the systems are designed in a way such that a collision of trains is physically impossible. Therefore, reliability of these systems usually is very high, with failures being caused by mechanical fatigue rather than by software errors.

Positioning with respect to the layers in our reference autonomy framework All modern rail vehicles have a Train Control and Management System (TCMS), which in conjunction with the on-board signalling computer (OBC) is responsible for the basic operations of the train. With respect to our three-level framework, TCMS and OBC realize the Reactions Layer. In the autonomous subway example described above, the Rules Layer is responsible for planning the journey, stops in the station, etc. A Principles Layer would be needed at least for unattended train operation, and even more for trains that calculate their own route on the tracks. There are many 'unusual' situations that can arise in such a scenario, which have to be dealt with by a Principles Layer.

Level of (future) autonomy Most present-day driverless trains, e.g., people-movers in airports, have almost no autonomy. They operate according to fixed time schedules, with fixed distances to other trains, and under tight supervision of a human operator. The situation becomes more interesting when looking at self-driving underground vehicles. Some of these trains can also drive on tracks used by conventional subway trains. Thus, the autonomous vehicle has to decide when to advance, and at which speed. In Nürnberg, such a system has been operating for over a decade, with more than 99% punctuality and without any accidents. A main benefit of autonomous operation is that it allows for a higher train frequency than with human-driven trains. In underground train systems, the probability of an obstacle on the tracks is low. For trams, regional trains, or high-speed trains, this is different. People or animals may cross the tracks, and thunderstorms may blow trees onto the tracks. Therefore, one of the main tasks of a train driver is to supervise the journey and to brake if necessary. Other tasks include monitoring for unusual events, checking the equipment before the trip, and passenger handling. There is an increasing number of electronic driver advisory systems such as speed recommendation systems, driver information systems, and others. Currently, all these systems leave responsibility for the trip to the driver. However, there is no compelling argument that an automated system could not do this task, increasing the autonomy of the system even further. The scope of autonomy is not high, since the functionality of the system is constrained by its environment (running on tracks).

The norm IEC 62290-1:2014 (Railway applications – Urban guided transport management and command/control systems – Part 1: System principles and fundamental concepts) defines five grades of automation (GoA). These can be compared with the levels of autonomy given in Definition 1. The fourth grade is driverless train operation (DTO), where the only tasks of the (human) train attendant are to handle passengers, control the doors, and to take over control in emergency situations. The fifth grade is unattended train operation (UTO), where no human operator from the train line is on board. That means the electronic systems need to be able to deal also with emergency situations.

Safety criticality (of autonomous aspects) Compared to other domains, railway homologation and operation is strictly regulated. For the approval of a particular system, the authorities rely on notified bodies, designated bodies, and associated bodies. These are institutions that assess the conformity of rail equipment to certain standards such as Cenelec 50128 [51]. For safety-critical software, known best practices, including formal methods, have to be applied.

Amount of available regulation (for autonomous aspects) The available regulation in the railway sector is not prepared to deal with higher levels of autonomy. For example, the movement authority, i.e., the permission to enter a certain block, is currently given by the roadside equipment or the railway control centre. There are research projects investigating whether it is possible to transfer this responsibility to the train; this would require safe localisation of a train and reliable communication between trains. In such a truly autonomous setting, the 'electronic driver' not only has to ensure that the track is free from obstacles, but also that there are no other trains in its way. Currently, it is unclear how such a system could be officially approved.

Suitable verification, validation, and analysis techniques Railway software must be extensively tested and verified before it can be put into use. Due to the cost and complexity, formal verification is reserved for functions of the highest criticality only. These are mainly the signalling and breaking functions. Testing is done on unit, integration, and system level: in contrast to the automotive domain, road tests are extremely expensive; thus, developers strive to limit the necessary road tests to a minimum. For autonomous driving, the problem becomes dramatically more severe, since it is not possible to test millions of different situations on the track.

4.4 Unmanned aerial system

Functionalities The term *Unmanned Aerial Systems* (UAS) covers a wide range of airborne vehicles, from toy quadcopters to military drones and autonomous missiles.

UAS already transport some goods and materials; challenges such as Urban Air Mobility (UAM)[9] and NASA's Advanced Air Mobility National Campaign (AAM)[10] promise to expand automated passenger and cargo transportation in urban, suburban, rural, and regional environments. These challenges must assuage public concerns that a falling UAS might harm people or property on the ground.

Positioning with respect to the layers in our reference autonomy framework The mapping to our three layer framework is straightforward, as is the need for all three layers. Rapid responses to the physical environment fall in the Reactions Layer, and UAS need to be able to apply rules to usual situations (Rules Layer) as well as deal with unanticipated situations by applying principles (Principles Layer).

Level of (future) autonomy Already, human pilots have a wealth of support systems at hand: navigation systems, autopilots, traffic collision avoidance systems, etc. Vital flight information gets recorded and transmitted to ground control. Cameras provide visual images from the cockpit. Therefore, moving from a human pilot within an air vehicle, to a human pilot remotely controlling the vehicle, and then towards the vehicle controlling itself autonomously, is a very popular approach. Yet it is fraught with difficulties (as discussed in earlier sections, in particular Sect. 2). In the near future we expect the scope of autonomy to be somewhat limited (medium), whereas it is possible that longer-term we will see UASs with higher levels of autonomous functionality.

Safety criticality (of autonomous aspects) Even though we are not (yet) considering UAS that carry human passengers, the safety criticality of UAS is very high. This is for the simple reason that a significant malfunction might lead to a collision, either between the UAS and a ground object (e.g., building, people), or between a UAS and another aircraft (potentially carrying passengers).

Amount of available regulation (for autonomous aspects) To provide a practical route to certification, including for non-autonomous systems, a range of documents/standards have been provided [193], perhaps the most relevant being the FAA documents *Software Considerations in Airborne Systems and Equipment Certification* [206, 210], *Formal Methods Supplement to DO-178C and DO-278A* [209], and *Design Assurance Guidance*

[9] https://www.faa.gov/uas/advanced_operations/urban_air_mobility/

[10] https://www.nasa.gov/aamnationalcampaign

for Airborne Electronic Hardware [208]. These standards provide directions toward the use of formal methods in the certification of (traditional) air systems, but say relatively little about autonomous (or even semi-autonomous) operation. The FAA has recently published official rules for non-hobbyist small UAS operations, *Part 107 of the Federal Aviation Regulations* [87] that certifies remote human operators for a broad spectrum of commercial uses of UAS weighing less than 25 kg.

Suitable verification, validation, and analysis techniques One approach, discussed in Sect. 3.3, to the analysis and potential certification of unmanned, particularly autonomous, air systems is to begin to show that the software replacing the pilot's capabilities effectively achieves the levels of competence required of a human pilot. For example, if we replace the flying capabilities of a pilot by an *autopilot* software component, then we must prove the abilities of the autopilot in a wide range of realistic flight scenarios. While this corresponds to the operational/control Reactions Layer within our approach, we can also assess decision-making against the required rules that UAS should follow. Pilots are required to pass many tests before being declared competent, with one such being knowledge of the 'Rules of the Air'. In [247] the agent making the high-level decisions in an UAS is formally verified against (a subset of) the 'Rules of the Air'. This type of analysis of the system's Rules Layer is necessary, though not sufficient, to provide broader confidence in autonomous air systems and the decision engines directing them. Model checking has successfully pinpointed unexpected emergent behaviours during design time in automated air traffic control (ATM) systems meant for organizing commercial aircraft [98, 183, 262]; its probabilistic variant provides useful analysis for making air traffic control design decisions [263]. These techniques are also ripe for UAS Traffic Management (UTM) verification. Human pilots possess strong elements of 'airmanship' and the ability and experience to cope with unexpected and anomalous situations. The Principles Layer must capture this, more complex, decision-making. Runtime verification on-board UAS (as well as in systems interacting with them, such as UTM or ground-monitoring systems) will be essential to assuring this aspect; see [50] for an initial UTM configuration embedding RV on-board, on-ground, and in the UTM cloud architecture. Runtime verification has also successfully identified security threats to UAS in-flight [99]. An initial attempt at the formalisation and verification of ethical decisions in unexpected situations appears in [70] wherein a simple ethical ordering of outcomes was used to cope with situations the UAS had no expectations of, and rules about.

4.5 Self-driving car

Functionalities Currently, the development of self-driving cars is one of the main innovation factors (besides e-mobility) in the automotive industry. Advanced driver assistance functions such as adaptive cruise control, lane keeping, or parking, are available in many cars. We can achieve partial autonomous driving (e.g., driving on the highway for a limited time) by combining such driver assistance systems. Although they are not yet available to all customers, cars with conditional autonomy (e.g., driving on the highway for an extended distance or autonomous parking in a parking deck) are already state of the art. Every major manufacturer has concrete plans to bring highly automated cars to the market in the next few years, although there is some debate as to whether these plans are feasible. Intermediate steps towards increasing autonomy include vehicle platooning/convoying whereby vehicles commit to being part of a (linear) formation and cede control of speed/direction to the platoon/convoy leader [27, 224, 232].

Positioning with respect to the layers in our reference autonomy framework In terms of our three-level categorisation, current 'driverless' cars essentially only have the lowest operational/control Reactions Layer to handle *lane following*, *speed control*, etc, while the human driver is responsible for following the rules and regulations concerning road use and for coping with unexpected situations or failures.

Once we move to the platoon/convoy (or 'road train') concept, responsibility for acting in accordance with more of the (developing) regulations [200] shifts to the autonomous system. Again, the human driver will be responsible for unexpected situations and failures, and will also take over responsibility for road use once the vehicle leaves the platoon/convoy. Thus, in this case, a Rules Layer is required.

However, the fully autonomous vehicles that we expect in the future will require software to be able to handle not just operational aspects such as steering, braking, parking, etc, but also comprehensive regulations such as the 'rules of the road'. Further, the Principles Layer of our autonomous system must conform to some high-level requirements concerning unexpected situations. Of course standards for these do not exist yet, but we might expect them to have some straightforward principles such as "*avoid harm to humans where possible*" and " *if a problem occurs, safely move to an approved parking area*." Many more principles/properties will surely be needed.

Level of (future) autonomy A long-term vision is for self-driving cars that are available on demand. People in need of a transport service could call an autonomous taxi via a smartphone app. Thus, there is no necessity to buy and maintain one's own car; resources are shared between many users. For this vision to become a reality, cars must be able to drive fully autonomously to any desired destination in all traffic situations, and must moreover be able to collaborate with the rest of the fleet to deliver an optimal service. As this vision is gradually realised, self-driving cars will have increasing scope of autonomy (eventually very high), but in the nearer term we expect more limited scope of autonomy.

Safety criticality (of autonomous aspects) Cars have a high level of safety criticality. The development of automotive software must follow functional safety standards, such as ISO 26262. This standard also prescribes safety requirements for design, development, and validation. For the verification and validation of software functionalities on the Reactions Layer and Rules Layer, we regard the existing techniques and regulations as being sufficient.

Amount of available regulation (for autonomous aspects) While there is considerable hype suggesting that autonomous vehicles are imminent, neither the automotive industry nor the regulators are at the point of being able to certify their reliability or safety. Global legal constraints, such as the Vienna Convention on Road Traffic [74], ensure that there must always be a person who can control the safe operation of the vehicle. There is a current political discussion on how to amend these regulations. A milestone in this direction was reached in 2020 when 60 countries adopted a United Nations Regulation that will allow for the safe introduction of automated vehicles in *certain* traffic environments. The UN Regulation, which enters into force in January 2021, establishes strict requirements for Automated Lane Keeping Systems (ALKS) in passenger cars that, once activated, are in primary control of the vehicle. However, the driver can override such systems and can be requested by the system to intervene, at any moment [256]. Previously, the US Department of Transportation issued a 'Federal Automated Vehicles Policy' [241]. Its 15-Point Safety Assessment includes 'Testing, validation, and verification of any HAV [Highly Automated Vehicles] system'. For SAE levels 4 and 5, if the driver has handed over control to the vehicle, liability rests with the manufacturer. Thus, most car manufacturers follow a 'trip recorder' approach: a black box specially protected against tampering collects all relevant data, and can be used to assign liability in case of an accident. In current legislation, liability depends on factors such as human reaction times or weather conditions. It is an open question how, or whether, these factors can transfer to the case where the driver is an autonomous agent instead of a human.

Suitable verification, validation, and analysis techniques We can see how the operational/practical constraints will be derived from many of the existing requirements on *cruise control*, *lane following*, etc, while we will see refined requirements for the sensing/monitoring capabilities within the vehicle. Verifying all of these aspects often involves a suite of (acceptable) testing schemes, such as requirements-driven test-case generation [240]. Just as the agent controlling an autonomous air vehicle can be verified with respect to the 'Rules of the Air' (see above), the agent controlling an autonomous road vehicle can be verified against the 'rules of the road' [6]. As with verification of UAS, model checking can straightforwardly check requirements for the Reactions Layer and Rules Layer during system design time; runtime verification provides required checks that the vehicle upholds these requirements on the road [93]. There has also been considerable work in using theorem proving to verify automated systems for vehicle control that carry through to higher-level decisions of the Principles Layer [96].

When it comes to the functionalities of the Principles Layer, the rules that a human driver is expected to follow need to be transformed into requirements suitable for an autonomous agent. This includes sensor/camera reliability and confidence in obstacle/sign/pedestrian recognition, and will also feed in to risk analysis component within the agent. Examples of Principles Layer rules include *"if there is low perceived risk of collision, continue at the highest admissible speed"*, and *"if there is high perceived risk of collision, or low confidence in the perceptions, then slow down"*. In fully autonomous vehicles, the reliability of the Principles Layer is a crucial aspect. It must be able to cope with road construction, failures, aggressive road users, and other unexpected events. Thus, the extraction of suitable requirements and their validation is highly important.

4.6 Personal assistant agent

Functionalities The now-ubiquitous personal assistant agent has origins earlier than the web [108, 110] and became mainstream with Apple's *Siri* in 2010 [28], and continued via Amazon Alexa, Microsoft Cortana, and Google Home [164]. The vision for such a *PAA* agent is a proactive, personalised, contextual assistance for a human user [259]. A PAA agent is a software entity, although the services it connects to and can command may include physical services such as smart home devices. So far, PAAs are mainly voice-controlled assistants that react to user requests such as looking for information on the web, checking the weather, setting alarms, managing lights, and automating thermostats in a smart home. Some apps classified as smart personal assistants connect to service providers, banks, social networks; 24me is one among them: according to its description on the web [1], it "automatically tells you what you need to do and also helps you take care of your things from within the app." Robotic pets such as the Paro seal can serve multiple roles, even contributing to healthcare; Paro robot harp seals use AI to move and make noises to learn to trigger dementia patients to pet them, thus relieving stress in the humans [245]. This technology extends to human emulation, and Gatebox's 'Azuma Hikari' system[11] promises a startling development:

[11] www.gatebox.ai/en/hikari

> *The Comforting Bride: Azuma Hikari is a bride character who develops over time, and helps you relax after a hard day. Through day-to-day conversations with Hikari, and her day-to-day behaviour, you will be able to enjoy a lifestyle that is more relaxed.*

Positioning with respect to the layers in our reference autonomy framework The Reactions Layer for a PAA consists of the PAA connecting to local and web services. The services can affect the world and the user's day-to-day life in both smaller and larger ways: from opening an app on a mobile device, to reserving a table at a restaurant, to changing the temperature in the user's house, to making a bank payment from her account. The Rules Layer, currently only realized in sophisticated systems, is responsible for the combination of such services, e.g., according to user preferences ("one hour after the scheduled dinner reservation, begin heating the house"). The Principles Layer would be not only responsible for exceptional and emergency circumstances, e.g., calling help when the human needs it, but also for giving advice in personal matters, e.g., whether to sign an insurance contract or not. The 'Azuma Hikari' character has the potential for a wide array of autonomous actions, up to Principles Layer behaviours.

Level of (future) autonomy The level and scope of autonomy are both limited so far, but we see a huge potential for PAAs to become increasingly autonomous, implementing Negroponte's vision of the agent that *"answers the phone, recognizes the callers, disturbs you when appropriate, and may even tell a white lie on your behalf. [...] If you have somebody who knows you well and shares much of your information, that person can act on your behalf very effectively. If your secretary falls ill, it would make no difference if the temping agency could send you Albert Einstein. This issue is not about IQ. It is shared knowledge and the practice of using it in your best interests"* [191].

Safety criticality (of autonomous aspects) Depending on what the PAA can control, it may be a safety-critical system: if it can make financial transactions, it might unintentionally cause economic loss to its user; if it can send messages on behalf of, or even pretend to be its user, it might cause severe damage to professional and personal relationships. Indeed, if the PAA is given control over aspects of the user's physical environment (such as locks on their home's doors), the possibilities for harm become more significant. While not safety-critical, the 'Azuma Hikari' assistant clearly has some potentially ethical issues [198] as captured in the BS8611, including anthropomorphism. Similar questions arise for the Paro seal [230]. If we extend our view of safety beyond physical harms to more general psychological/ethical harms then there are issues here.

Amount of available regulation (for autonomous aspects) To our knowledge, there is no regulation of PAAs, other than general consumer electronics regulation and regulations in specific sectors that co-incidentally apply to the PAA's services such as regulations concerning protocols for accessing banking institution services. While there is no single data protection legislation in the US, in Europe the General Data Protection Regulation (GDPR [80]) became operative in May 2018. GDPR regulates the processing by an individual, a company, or an organisation of personal data relating to individuals in the EU. A PAA based in Europe should comply to the GDPR. As an example, a PAA should not share information by making unauthorized calls on behalf of its user and should not share sensitive data (pictures, logs of conversations, amount of money in the bank account, health records) of its user and his or her relatives. It should also govern personal data storage to ensure an acceptable level of protection. On the other hand, a PAA should be allowed to override privacy laws and call emergency services if it suspects its user is unconscious from an accident, suicidal, or otherwise unable to ask for help, and it should share health information with the first aid personnel. Relating to the ethical/psychological issues above, standards such as BS8611 and the IEEE's P7000 series are relevant.

Suitable verification, validation, and analysis techniques Like any complex piece of software, the PAA should undergo a careful formal verification and testing program before release. However, we believe that the most suitable technique to prevent it from unsafe behaviour, particularly if the system is adaptive, is runtime verification. Financial transactions might be monitored at runtime, and blocked as soon as their amount, or the amount of the exchanged message, overcome a given threshold. Calls to unknown numbers should not necessarily be blocked, as they might be needed in some emergency cases, but they should be intercepted and immediately reported to the user. Commands to smart devices should be prevented, if they belong to a sequence of commands meeting some dangerous pattern.

4.7 Home assistant robot

The situation changes if we consider home assistant robots that are not only performing simple tasks, but also issuing medical reminders, providing simple physiotherapeutic treatment, and even offering 'companionship'. Consider a robot such as the 'Care-o-Bot'.[12] A number of domestic assistants are already available. These span a range, from the above Robot Vacuum Cleaner right up to Care-o-Bot systems, and similar systems, such as Toyota's Human-Support Robot Family.[13] We will primarily consider the more sophisticated end of this spectrum.

Functionalities: Robots such as Care-o-Bot are intended to be a general domestic assistant. They can carry out menial cleaning/tidying tasks, but can also (potentially, given the regulatory approval) fetch deliveries from outside the home and provide food and drink for the occupant(s) of the house. (There are also specialized robots for some of these tasks, such as the University of Texas at Dallas' meal-delivering robots.[14]) Each such robot is typically mobile (e.g., wheels) and has at least one manipulator, usually an 'arm'. Yet this type of robot can potentially go further, becoming a true *social robot*, interacting with the occupant(s) (e.g., what to get for lunch?), and engaging in social dialogue (e.g., favourite TV show?). These robots (again, with appropriate approval) also have the potential to measure and assess physical and psychological health. Clearly, with this wide range of functionalities and increasing autonomy, robots of this form should be both highly-regulated and will have many, non-trivial, decisions to make.

Positioning with respect to the layers in our reference autonomy framework The need for all of our layers is clear. Any physical interaction with the environment, or movement within a house, will require a complex Reactions Layer. Similarly, one would expect the regulatory framework for such robotic systems to be quite complex and so, necessarily, the Rules Layer will also likely be comprehensive. In the future the combination of broad functionality, increasing autonomy, and responsibility for health and well-being of human occupants will surely lead to quite complex ethical issues [10, 53, 184, 223] and so to a truly non-trivial Principles Layer.

[12] www.care-o-bot.de

[13] www.toyota-global.com/innovation/partner_robot

[14] https://www.dallasnews.com/business/technology/2019/12/26/on-university-of-texas-at-dallas-growing-campus-meal- delivering-robots-make-splashy-debut/

Level of (future) autonomy: In principle, the scope and level of autonomy can both be high but, as noted below, the regulatory frameworks limit this at present. In the future, home care robots might become much more than service providers. Especially with people who are isolated and find little human contact, these robots might begin to guide and accompany their owners, remind them to take their medicine or drink water, and even provide simple social companionship (for example, asking about relatives, about the news, or about TV programmes seen recently). Especially in these latter cases, the issue of *trust* is crucial [7, 25, 222], since no one will use a robot in this way if they do not trust the robot. It becomes a central issue for the Principles Layer to assess how much to build trust and how much to stay within legal/regulatory bounds. Doing what the owner wants the robot to do can build trust, whereas refusing to do something because of minor illegality erodes trust very quickly [237].

Safety criticality (of autonomous aspects) In addition to the various issues that apply to Personal Assistant Agents, the broader role of Home Assistant Robots makes them more safety critical. For instance, if medicine is administered incorrectly, or if a Home Assistant Robot failures to raise the alarm when a human has an adverse health event, then lives may be put at risk. Therefore, it appears that clear regulations specifically developed for such robots will be needed, alongside standards targeting exactly the issues concerning these installations. With the complex practical, social, and health-care potential, the range of standards needs to go beyond physical safety and on to issues such as ‘ethical risk’ [43].

Amount of available regulation (for autonomous aspects): There appears to be very little current regulation specifically designed for autonomous robotics, let alone such autonomous domestic assistants. While there are some developing standards that have relevance, such as ISO 13482 [153], the regulatory oversight for domestic robotic assistants is unclear. In some cases they are treated as industrial robots, in some cases as medical devices (especially if health monitoring is involved), and in others they are straightforwardly treated as electro-mechanical devices. In some jurisdictions (e.g., the UK) responsibility for regulation falls to the local regulatory authority, whereas in others regulatory oversight is a national responsibility. Furthermore, the standards that are appealed to are not specific standards for autonomous domestic robots, but rather standards for software, robots, or medical appliances.

On particular challenge concerns learning new behaviours. These might be specified by the user or might result from analytical analysis by the robot. These learned behaviours might well be ‘new’, i.e., not part of the robot’s behaviour when initially deployed. In this case, there should be some mechanism to assess whether the new, learned behaviour is consistent with the prescribed regulations. However, this assessment must either take place offline, in which case there will be some ‘down time’ for the robot, or online, in which case we will need effective and efficient techniques for carrying out this assessment. In the case where the human has specified/requested some new behaviour that is ‘in conflict’ with the regulations, the Principles Layer becomes essential as it must make decisions about whether to employ, or ignore, this new behaviour.

Suitable verification, validation, and analysis techniques Ignoring the analysis of standard robot manipulation and movement aspects, there has been specific work on the analysis of Reactions Layer [165, 253] and Rules Layer [71, 246] issues. While runtime verification is, in principle, vital for continuously ensuring assistive robots’ safe operation, it is hugely challenging to embed this potentially computationally-heavy capability on-board these robot’s nimble architectures, and to do so in an unobtrusive way, that does not impact their power, weight, timing guarantees, or other vitals. A pioneering case study embedded the R2U2 runtime verification engine on-board the FPGA controlling the knee of Robonaut2, a humanoid robot purposed with assisting astronauts on the International Space Station [163]. As we would expect, the ethical issues concerning such (potential) robots are of great interest to both philosophers and formal methods [11, 25]. Interestingly, issues such as privacy and security are only gradually being considered [257].

5 Future challenges

This section looks ahead. Based on the discussion in previous sections, it defines a number of challenges. We break these down into three areas, depending on the intended audience, i.e., who are the people who could make progress on the challenge? Specifically, we consider challenges to researchers, to engineers, and to regulators.

5.1 Research challenges

There are a number of challenges that we identify for the research community. One group of challenges relates to the overarching question of how to extend from current verification settings and assumptions, to the settings and assumptions required? Specific questions include:

- As Sections 3 and 4 highlight, in various domains autonomous systems need to perform ethical reasoning from our Principles Layer. However, how to specify the reasoning required, and perform it effectively, is still an open question, as is how to specify verification properties that relate to such reasoning.
- In domains that involve interaction with humans, especially close interaction, such as human-agent teams, there is a need for ways to elicit and model the capabilities and attitudes (e.g., goals, beliefs) of the humans. How can we extend both formal specification and formal verification to effectively reason about evolving knowledge? Extending specification and verification techniques like model checking to logics like Temporal Epistemic Logic, which reasons about the evolution of knowledge over time, is an challenging research area [54].
- Once we figure out how to effectively specify properties of systems at each of the three layers of autonomy, we are faced with overcoming the specification bottleneck [218]. How to we (semi-) automate these specification processes so that they are not onerous but can be integrated into design and development processes? How do we measure specification quality? How do we organize specifications in a transparent hierarchy to enable cross-checking with regulations?
- Since autonomous systems, even in safety-critical situations, are increasingly likely to make some use of machine learning, how can we verify such systems? If a system is able to learn at runtime, then some sort of runtime verification will be needed, although there may also be a role for mechanisms that limit the scope of the learning in order to ensure that important desired properties cannot be violated. Limits imposed, e.g., by runtime verification are more straightforward for behaviours in the Reactions Layer but learned decisions in the Rules Layer prove a greater challenge to limit via predefined boundaries and we do not yet have a framework for bridging such a strategy to the Principles Layer.
- How can we handle larger and more complex systems (scalability of verification methods)? This includes systems with potentially not just multiple but *many* agents. Verify-

ing distributed swarms at the Reactions Layer already presents a scalability challenge. Such systems will require both system (or swarm-level) and sub-system (or agent-level) requirements at the Rules Layer and the Principles Layer. How do we capture these into a transparent, certifiable organisational structure?
- Finally, how we can improve the way that we design and build systems to make them more amenable to verification? One possible approach is to use *synthesis* to build systems that are provably correct from verification artefacts. Part of the research challenge is to develop techniques like synthesis in ways that are transparent and can be integrated into industrial culture; automatically generating safety-critical autonomous systems via an automated system that comes across as some sort of black magic will face major challenges to use and adaptation, especially if the generated autonomous system is capable of Principles Layer decisions.

In addition to challenges relating to extending verification to richer, and more complex, settings, there are also a number of research challenges that relate to the broader social context of verification. Perhaps most fundamentally, we need better methods for systematically identifying (i.e., deriving) properties to be verified. Sect. 3.3) sketches an outline of such a process, but more work is required to flesh this out, and refine it through iterated use and evaluation. One particular challenge is dealing with unusual situations that are particularly hard to anticipate. Another area that poses research challenges is the possibility of using natural language processing to create structured formal (or semi-formal) requirements from textual artefacts, which we discussed in Sect. 2.3.1.

There is also a big research challenge to develop ways to engineer systems that are transparent, and, in particular, systems that can explain themselves to users, regulators, and bystanders. This is especially tricky when the systems need to explain Principles Layer decisions [211]. While there is the whole subfield of explainable AI (XAI), here we highlight the particular needs of safety-critical autonomous systems – where explainability might need to focus less on explaining specific individual behaviours to users, and more on explaining how a system operates to regulators.

5.2 Engineering challenges

We now turn to engineering challenges. These are challenges that need to be addressed to practically engineer reliable autonomous systems, once the underlying research challenges have been adequately addressed.

- An existing challenge that is exacerbated by increasing levels of autonomy involves the need to track assumptions and their provenance, and to manage the system as it evolves (maintenance). If requirements change, and the system is modified, how can we update the verification, as well as manuals and other documentation? This question may be different for different layers of autonomy, since we may be able to engineer (semi-)automated solutions for updates to the Reactions Layer via straightforward checks on numerical bounds, but actions in the Principles Layer are more often the result of complex emergent behaviours that would be more difficult to maintain. How do we know, e.g., if societal expectations are changing and what that looks like as a system-wide, high-level philosophical update?
- We identified research challenges relating to extending the scope and setting to deal with ethical reasoning from our Principles Layer and to deal with machine learning,

which can be applied at any level of autonomy. Systems that perform ethical reasoning must be built; building systems that involve learning also has engineering challenges.
- A research challenge that we identified was how to model humans, for example in the context of human-agent teamwork. There is a related engineering challenge that crosscuts all levels of autonomy: how to design systems that 'work well' with humans. Note this is a design challenge, not a research challenge (there is a body of e.g., HCI research on this).
- Another research challenge was using *synthesis* to create provably correct systems. A related engineering challenge is managing this process. In practical terms, even if we have technology that can generate a system (or parts of a system) from verification artefacts, there are engineering challenges in managing this process, and having a metasystem where the user clicks 'go' and the system is built from V&V artefacts. Once we meet this engineering challenge, we face the additional challenge of how to update synthesized systems once they are in operation: if the system was generated from provably correct artifacts, then modified, what does proper maintenance look like?
- We noted above a number of research challenges that relate to the broader social context. There are also engineering challenges that relate to this. The big challenge is linking verification of design artefacts to the broader needs of stakeholders, in particular regulators. Suppose that a software system for controlling a train is verified. How can this be used to provide regulators with a convincing safety argument? One particular aspect is trust in the process and tools: how do we know that the verification tools are themselves correct? See the discussion of regulator challenges below.
- Finally, just as there are research challenges in scaling to handle larger and more complex systems, so too there are engineering challenges that relate to verifying large and complex systems. These concern techniques for making effective use of computing resources (be they traditional high-performance computing, or cloud computing) in order to effectively verify systems, and manage this process. Engineering optimisation problems include how to leverage increasingly parallel architectures and efficiently manage load balancing.

Along with these challenges, we mention the difficulty that companies can have in finding qualified engineering staff [181].

5.3 Regulatory challenges

Finally, there are challenges for regulators. A number of challenges relate to the existence of multiple stakeholders, who may be in a complex state that combines cooperation (e.g., to create standards) and competition. Related challenges to this include: how to manage disclosure of information that is sensitive (e.g., industrial 'secrets'), and how to reach consensus of involved stakeholders, e.g., car OEMs. This is particularly important if autonomous systems from different manufacturers have to collaborate, e.g., in vehicle platooning.

Regulators face a major challenge with respect to how to define and regulate common interfaces for compositional verification, especially for autonomous systems like self-driving cars or UAS, so that the low-level structures of each agent can remain proprietary but the agents can interact with each other and with an automated environment, such as the UAS Traffic Management (UTM) system or a smart road, in a way that is certifiably safe. This requires a deeper understanding of the technical engineering and verification processes than has previously been required of regulatory bodies and a need for regulatory bodies to be more responsive to evolving architectures. This will be a challenge for regulating Reactions Layer autonomy; it is even more complicated for higher-level autonomy.

More broadly, the regulatory landscape is complex, with multiple actors (governments, courts, companies, etc), which poses challenges around how to manage differences between jurisdictions, or between regulators with overlapping domains of interest. If autonomous systems operate in different countries, they should comply with all national regulations of the involved countries. However, the legal requirements may differ, and even contradict each other. Here, multi-national agreements and treaties are necessary for regulation of Rules Layer autonomy.

When we consider any form of Principles Layer ethical reasoning done by autonomous systems, then there is a challenge in how to obtain sufficient agreement amongst various stakeholders (e.g., government, civil society, manufacturers) about the specification of what is appropriate ethical behaviour, or even a clear delineation distinguishing between behaviour that is clearly considered ethical, behaviour that is clearly considered unethical, and behaviour where there is not a clear consensus, or where it depends on the underlying ethical principles and framework.

A related point is the legal notion of responsibility, which is a question for regulators, and for society more broadly in the form of governments and legal systems; see for instance [101] for an introduction. This question is an opportunity for regulators to drive the research: what do verification engines need to be able to prove or certify about an autonomous system in order to support liability?

Finally, as mentioned in Sects. 4 and 5.2 , there are challenges in trusting tools and processes themselves. How can the verification tools be trusted? [229] To what extent can they be verified? [212]. There are currently some standards for verification frameworks used to certify autonomous systems, like the NASA Software Engineering Requirements Appendix D. Software Classifications[15], but regulators will need to question whether there needs to be more fine-grained classifications of such tools going forward rather than NASA's Class C encompassing all 'software used to verify system requirements'. Should there be different classifications for verification tools addressing Reactions Layer requirements from those evaluating Principles Layer requirements?

6 Conclusion

Autonomous systems have great potential to transform our world. The substantial adoption of even just one type of autonomous system – self-driving cars – will substantially alter passenger transport and the geography of cities, e.g., enhancing mobility for those who cannot drive, and reducing the need for parking spaces. More fundamentally, autonomous systems change our relationship with technology, as technology and human society reach an "intimate entanglement" [94]. However, ensuring that autonomous systems are fit-for-purpose, especially when their function is in any way safety-critical, is crucial for their adoption. This article therefore addresses the fundamental question: "*How can the reliability of an autonomous software system be ensured?*"

In order to pave the way towards an answer to this question, we identify four main objectives for our research in Sect. 1.2: 1) proposing a framework for viewing

[15] https://nodis3.gsfc.nasa.gov/displayDir.cfm?Internal_ID=N_PR_7150_002C_&page_name=AppendixD

autonomous systems in terms of three layers; 2) showing that this framework is general; 3) discussing how certification/regulation might be achieved; and 4) identifying challenges and future work for our intended audience.

We reached these four objectives by:

1. reviewing the state of the art, including standards, existing approaches for certification, and open issues (Sect. 2), in order to propose a way forward towards a framework for certification of safety-critical autonomous systems firmly based on the state-of-the-art literature; our new three-layer framework provides a basis for structuring such systems in Sect.3.1 where each layer has identifiable challenges, expectations, and examples;
2. illustrating applications of the proposed framework on a range of diverse systems, in a range of domains (Sect. 4);
3. indicating what is needed from regulators, and outlining a process for identifying requirements in Sect. 3.2 and 3.3 ; given the importance of verifying autonomous systems, we survey a range of verification techniques, considering their applicability, strengths, and weaknesses with respect to autonomous systems (Sect. 3.4); and
4. articulating a range of challenges and future work, including challenges to regulators, to researchers, and to developers in Sect. 5.

We anticipate the value of this study as a reference to all three groups, researchers, engineers, and regulators, going forward. In addition to the specific challenges we pose here, there are also a number of other questions arising from the emergence of autonomous systems. These are cross-cutting, in that they pose challenges to all of the key stakeholders (researchers, engineers, and regulators) – as well as to philosophers, psychologists, and sociologists for their sometimes abstract and speculative nature, and for the societal and psychological impact of having autonomous systems interact with humans in their daily life. These questions include:

- How to deal with tacit knowledge, e.g., where humans learn by ‘feel’ (e.g., pilots)? Should each individual autonomous system learn, or should the knowledge evolve in a population?
- How to deal with security challenges, in particular, if the autonomous system cannot be continually supervised? Should an autonomous system have ‘self-defence’ capabilities against humans?
- How to deal with Quality of Service (QoS) requirements, in particular, for a large group of autonomous systems and many users?
- How to deal with varying interpretations, inconsistent requirements, and missing context?
- How to deal with contradicting requirements from different stakeholders? Is there a notion of ‘loyalty’ for autonomous systems?
- How to deal with changing standards, attitudes and morals, as well as evolving societal expectations and roles?
- How to properly deal with the emerging issue of autonomous systems’ self-awareness [67, 73, 225]?
- How to deal with the complexity of ‘fractal autonomy’, where (autonomous) systems consist of autonomous sub-systems? Can systems be defined as autonomous just because their sub-systems are?

- How to shine light on unstated assumptions, which may look different at each of our three layers of autonomy, and pose challenges for certification. This is related to the question of how to identify and react to bias, both when the bias is potentially harmful (e.g., bias against one group of human users), and when it poses as an unstated assumption vital to certification (e.g., the system design relies on a common human bias toward reacting a certain way).

We hope that the initial framework, and the specific challenges, can form a map in guiding the various communities along the path towards a framework for certification of reliable autonomous systems.

Acknowledgements We thank the organisers and participants of the Dagstuhl 19112 workshop, and the anonymous reviewers of this article. Thanks to Simone Ancona for the drawings in Sect. 1. MF was partially supported by a Royal Academy of Engineering *Chair in Emerging Technologies*, and by EPSRC grants S4 (EP/N007565), RAIN (EP/R026084), ORCA (EP/R026173), FAIR-SPACE (EP/R026092) and Verifiability Node (EP/V026801). KYR was partially supported by NSF CAREER Award CNS-1552934, NASA ECF grant NNX16AR57G and NSF PFI:BIC grant CNS-1257011. NYS was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under grant 952215.

References

1. 24me Company. 24me Smart Personal Assistant. URL https://www.twentyfour.me/.
2. Abate, A., Katoen, J.-P., & Mereacre, A. (2011). Quantitative automata model checking of autonomous stochastic hybrid systems. In *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control (HSCC)* (pp. 83–92). ACM.
3. Adolf, F.-M., Faymonville, P., Finkbeiner, B., Schirmer, S., & Torens, C. (2017). Stream runtime monitoring on UAS. In *Proceedings of International Conference on Runtime Verification* (pp. 33–49).
4. Alexander, R., Hall-May, M., & Kelly, T. (2007). Certification of autonomous systems. In *Proceedings of 2nd Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre (DTC) Annual Technical Conference*.
5. Alur, R., Henzinger, T. A., Lafferriere, G., & Pappas, G. J. (2000). Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, *88*(7), 971–984.
6. Alves, G. V., Dennis, L., Fernandes, L., & Fisher, M. (2020). Reliable Decision-Making in Autonomous Vehicles. In A. Leitner, D. Watzenig, & J. Ibanez-Guzman (Eds.), *Validation and verification of automated systems: Results of the ENABLE-S3 Project* (pp. 105–117). Cham: Springer, New york.
7. Amirabdollahian, F., Dautenhahn, K., Dixon, C., Eder, K., Fisher, M., Koay, K. L., Magid, E., Pipe, A., Salem, M., Saunders, J., & Webster, M. (2013). Can You Trust Your Robotic Assistant? In *International Conference on Social Robotics*, volume 8239 of *LNCS* (pp. 571–573). Springer.
8. Ancona, D., Ferrando, A., & Mascardi, V. (2016). Comparing trace expressions and Linear Temporal Logic for runtime verification. In *Theory and Practice of Formal Methods*, (pp. 47–64). Springer, New york
9. Ancona, D., Ferrando, A., & Mascardi, V. (2017). Parametric runtime verification of multiagent systems. *AAMAS*, *17*, 1457–1459.
10. Anderson, M., & Anderson, S. L. (2008). EthEl: Toward a principled ethical eldercare robot. In *Proc. AAAI Fall Symposium on AI in Eldercare: New Solutions to Old Problems*.

11. Anderson, M., & Anderson, S. L. (2011). *Machine Ethics*. : Cambridge University Press.
12. Appel, K., & Haken, W. (1989). *Every Planar Map is Four-Colorable*, volume 98 of *Contemporary Mathematics*. Providence, RI: American Mathematical Society. ISBN 0-8218-5103-9. https://doi.org/10.1090/conm/098.
13. Arcaini, P., Bonfanti, S., Gargantini, A., Mashkoor, A., & Riccobene, E. (2015). Formal validation and verification of a medical software critical component. In *13th ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2015, Austin, TX, USA, September 21-23, 2015* (pp. 80–89). IEEE. https://doi.org/10.1109/MEMCOD.2015.7340473.
14. Areias, C., Cunha, J. C., Iacono, D., & Rossi, F. (2014). Towards certification of automotive software. In *Proceedings of 25th IEEE International Symposium on Software Reliability Engineering Workshops ISSRE* (pp. 491–496). https://doi.org/10.1109/ISSREW.2014.54.
15. Arkin, R. C. (2008). Governing lethal behavior: embedding ethics in a hybrid deliberative/reactive robot architecture. In *Proceedings of 3rd ACM/IEEE international conference on Human Robot Interaction (HRI'08)* (pp. 121–128). https://doi.org/10.1145/1349822.1349839.
16. AV-TEST Institute. (2019). Robot vacuums undergo a security check: trustworthy helpers around the house or chatty cleaning appliances? URL https://www.av-test.org/en/news/robot-vacuums-undergo-a-security-check-trustworthy-helpers-around-the-house-or-chatty-cleaning-appli/. Archived from the original URL at: https://web.archive.org/web/20200613234231/https://www.av-test.org/en/news/robot-vacuums-undergo-a-security-check-trustworthy-helpers-around-the-house-or-chatty-cleaning-appli/.
17. Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press. ISBN 026202649X.
18. Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS One*, *12*(7), e0180944.
19. Bartocci, E., Bortolussi, L., Brázdil, T., Milios, D., & Sanguinetti, G. (2017). Policy learning in continuous-time markov decision processes using gaussian processes. *Perform. Eval.*, *116*, 84–100. https://doi.org/10.1016/j.peva.2017.08.007.
20. Basin, D. A., Klaedtke, F., Müller, S., & Pfitzmann, B. (2008). Runtime monitoring of metric first-order temporal properties. In *Proceedings of 28th IARCS Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'08)* (pp. 49–60). https://doi.org/10.4230/LIPIcs.FSTTCS.2008.1740.
21. Bauer, B., Müller, J. P., & Odell, J. (2000). Agent UML: A formalism for specifying multiagent software systems. In P. Ciancarini, & M. J. Wooldridge (Eds.), *Agent-Oriented Software Engineering, First International Workshop, AOSE 2000, Limerick, Ireland, June 10, 2000, Revised Papers*, volume 1957 of *Lecture Notes in Computer Science* (pp. 91–104). Springer. https://doi.org/10.1007/3-540-44564-1_6.
22. Beck, K. (2003). *Test-driven development: by example*. : Addison-Wesley Professional.
23. Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for agile software development. URL http://agilemanifesto.org/.
24. Bensalem, S., Ganesh, V., Lakhnech, Y., Munoz, C., Owre, S., Rueß, H., Rushby, J., Rusu, V., Saıdi, H., Shankar, N., et al. (2000). An overview of SAL. In *Proceedings of 5th NASA Langley Formal Methods Workshop*. Williamsburg, VA.
25. Bentzen, M., Lindner, F., Dennis, L., & Fisher, M. (2018). Moral Permissibility of Actions in Smart Home Systems. In *Proceedings of FLoC 2018 Workshop on Robots, Morality, and Trust through the Verification Lens*.
26. Benzel, T. (1984). Analysis of a kernel verification. In *Proceedings of 1984 IEEE Symposium on Security and Privacy* (pp. 125–133). https://doi.org/10.1109/SP.1984.10015.
27. Bergenhem, C., Huang, Q., Benmimoun, A., & Robinson, T. (2010). Challenges of platooning on public motorways. In *Proceedings of 17th World Congress on Intelligent Transport Systems* (pp. 1–12).
28. Berry, P. M., Gervasio, M. T., Peintner, B., & Yorke-Smith, N. (2011). PTIME: personalized assistance for calendaring. *ACM Trans. Intelligent Systems and Technology*, *2*(4), 40:1–40:22. https://doi.org/10.1145/1989734.1989744.
29. Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In L. C. Briand, & A. L. Wolf (Eds.), *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA* (pp. 85–103). IEEE Computer Society. https://doi.org/10.1109/FOSE.2007.25.
30. Biere, A., Heljanko, K., & Wieringa, S. (2011). AIGER 1.9 and beyond. *Available at* fmv.jku.at/hwmcc11/beyond1.pdf.
31. Birnbacher, D., & Birnbacher, W. (2017). Fully autonomous driving: Where technology and ethics meet. *IEEE Intelligent Systems*, *32*(5), 3–4. https://doi.org/10.1109/MIS.2017.3711644.

32. Bloomfield, R., & Bishop, P. (2010). Safety and assurance cases: Past, present and possible future - an adelard perspective. In C. Dale & T. Anderson (Eds.), *Making systems safer* (pp. 51–67). London, UK: Springer.
33. Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co. Inc. ISBN 0-201-57168-4.
34. Bordini, R. H., Fisher, M., Pardavila, C., & Wooldridge, M. J. (2003). Model checking AgentSpeak. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings* (pp. 409–416). ACM. https://doi.org/10.1145/860575.860641.
35. Bordini, R. H., Fisher, M., Visser, W., & Wooldridge, M. J. (2004). Model checking rational agents. *IEEE Intelligent Systems*, *19*(5), 46–52. https://doi.org/10.1109/MIS.2004.47.
36. Bordini, R. H., Fisher, M., Visser, W., & Wooldridge, M. J. (2006). Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, *12*(2), 239–256. https://doi.org/10.1007/s10458-006-5955-7.
37. Bozzano, M., Cimatti, A., Pires, A. F., Jones, D., Kimberly, G., Petri, T., Robinson, R., Tonetta, S. (2015). Formal design and safety analysis of air6110 wheel brake system. In *International Conference on Computer Aided Verification* (pp. 518–535). Springer.
38. Brat, G., & Venet, A. (2005). Precise and scalable static program analysis of NASA flight software. In *2005 IEEE Aerospace Conference*, (pp. 1–10). IEEE.
39. Brat, G., Navas, J. A., Shi, N., & Venet, A. (2014). IKOS: A framework for static analysis based on abstract interpretation. In *International Conference on Software Engineering and Formal Methods* (pp. 271–277). Springer.
40. Bremner, P., Dennis, L. A., Fisher, M., & Winfield, A. F. T. (2019). On Proactive, transparent, and verifiable ethical reasoning for robots. *Proceedings of the IEEE*, *107*(3), 541–561. https://doi.org/10.1109/JPROC.2019.2898267.
41. Bringsjord, S., Arkoudas, K., & Bello, P. (2006). Toward a general logicist methodology for engineering ethically correct robots. *IEEE Intelligent Systems*, *21*(4), 38–44.
42. British Standards Institution. BSI web site. URL https://www.bsigroup.com/.
43. British Standards Institution (BSI) (2016). BS 8611 – robots and robotic devices — guide to the ethical design and application. URL https://shop.bsigroup.com/ProductDetail/?pid=000000000030320089.
44. Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE J. Robotics Autom.*, *2*(1), 14–23. https://doi.org/10.1109/JRA.1986.1087032.
45. Butler, R. (1996). *An introduction to requirements capture using PVS: specification of a simple autopilot*. Technical report, NASA Langley Technical Report Server.
46. Butler, R. W., & Finelli, G. B. (1993). The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, *19*(1), 3–12.
47. Cambridge Academic Content Dictionary (2020). Definition of 'certification'. URL https://dictionary.cambridge.org/dictionary/english/certification.
48. Cambridge English Dictionary (2020). Definition of 'autopilot'. URL https://dictionary.cambridge.org/dictionary/english/autopilot.
49. Cambridge English Dictionary (2020). Definition of 'regulation'. URL https://dictionary.cambridge.org/dictionary/english/regulation.
50. Cauwels, M., Hammer, A., Hertz, B., Jones, P., & Rozier, K. Y. (September 2020). Integrating runtime verification into an automated uas traffic management system. In *Proceedings of DETECT: international workshop on moDeling, vErification and Testing of dEpendable CriTical systems*, Communications in Computer and Information Science (CCIS), page TBD, L'Aquila, Italy. Springer.
51. CENELEC (2011). CENELEC - EN 50128 – railway applications - communication, signalling and processing systems - software for railway control and protection systems. URL https://standards.globalspec.com/std/1678027/EN%2050128
52. Chapman, D. (1987). Planning for conjunctive goals. *Artif. Intell.*, *32*(3), 333–377. https://doi.org/10.1016/0004-3702(87)90092-0.
53. Charisi, V., Dennis, L., Fisher, M., Lieck, R., Matthias, A., Slavkovik, M., Sombetzki, J., Winfield, A. F. T., & Yampolskiy, R. (Mar. 2017). Towards moral autonomous systems. *ArXiv e-prints*.
54. Cimatti, A., Gario, M., & Tonetta, S. (2016). A lazy approach to temporal epistemic logic model checking. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems* (pp. 1218–1226).
55. Clarke, E. M., & Schlingloff, B.-H. (2001). Model Checking. In A. Robinson & A. Voronkov (Eds.), *Handbook of Automated Reasoning* (pp. 1635–1790). : Elsevier and MIT Press.

56. Clarke, E. M., Grumberg, O., & Peled, D. A. (2000). *Model Checking*. : The MIT Press. ISBN 0262032708.
57. Cobleigh, J. M., Giannakopoulou, D., & Păsăreanu, C. S. (2003). Learning assumptions for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 331–346). Springer, Newyork
58. Cofer, D. D., Hatcliff, J., Huhn, M., & Lawford, M. (2013). Software certification: Methods and tools (Dagstuhl seminar 13051). *Dagstuhl Reports*, *3*(1), 111–148. https://doi.org/10.4230/DagRep.3.1.111.
59. Cousot, P., Cousot, R., Feret, J., Miné, A., Rival, X., Blanchet, B., Monniaux, D., & Mauborgne, L. Astrée. URL http://www.astree.ens.fr/.
60. Currit, P. A., Dyer, M. G., & Mills, H. D. (1986). Certifying the reliability of software. *IEEE Trans. Software Eng.*, *12*(1), 3–11.
61. de Araújo, R. P., Mota, A. C., & Nogueira, S. d. C. (Aug 2017). Probabilistic analysis applied to cleaning robots. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)* (pp. 275–282). https://doi.org/10.1109/IRI.2017.61.
62. Dalpiaz, F., Ferrari, A., Franch, X., & Palomares, C. (2018). Natural language processing for requirements engineering: The best is yet to come. *IEEE Software*, *35*(5), 115–119. https://doi.org/10.1109/MS.2018.3571242.
63. Dastani, M., Torroni, P., & Yorke-Smith, N. (2018). Monitoring norms: A multi-disciplinary perspective. *Knowledge Eng. Review*, *33*, e25. https://doi.org/10.1017/S0269888918000267.
64. Dennett, D. C. (1989). *The Intentional Stance*. Cambridge, MA, USA: MIT Press.
65. Denney, E., & Pai, G. (2018). Tool support for assurance case development. *Autom. Softw. Eng.*, *25*(3), 435–499. https://doi.org/10.1007/s10515-017-0230-5.
66. Dennis, L. A. (2018). The MCAPL Framework including the Agent Infrastructure Layer and Agent Java Pathfinder. *The Journal of Open Source Software*, *3*(24)
67. Dennis, L. A., & Fisher, M. (2020). Verifiable self-aware agent-based autonomous systems. *Proceedings of the IEEE*, *108*(7), 1011–1026. https://doi.org/10.1109/JPROC.2020.2991262.
68. Dennis, L. A., Fisher, M., Webster, M., & Bordini, R. H. (2012). Model checking agent programming languages. *Automated Software Engineering*, *19*(1), 5–63.
69. Dennis, L. A., Fisher, M., Lincoln, N. K., Lisitsa, A., & Veres, S. M. (2016). Practical verification of decision-making in agent-based autonomous systems. *Automated Software Engineering*, *23*(3), 305–359. https://doi.org/10.1007/s10515-014-0168-9. ISSN 0928-8910.
70. Dennis, L. A., Fisher, M., Slavkovik, M., & Webster, M. (2016). Formal verification of ethical choices in autonomous systems. *Robotics and Autonomous Systems*, *77*, 1–14. https://doi.org/10.1016/j.robot.2015.11.012.
71. Dixon, C., Webster, M., Saunders, J., Fisher, M., & Dautenhahn, K. (2014). The Fridge Door is Open — Temporal Verification of a Robotic Assistant's Behaviours. In *Advances in Autonomous Robotics Systems (TAROS)*,volume *Lecture Notes in Computer Science* (pp. 97–108). Springer. Newyork
72. Dutilleul, S. C., Lecomte, T., & Romanovsky, A. B. (Eds.) (2019). *Proceedings of 3rd International Conference on Reliability, Safety, and Security of Railway Systems (RSSRail'19)*, volume 11495 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-030-18743-9. https://doi.org/10.1007/978-3-030-18744-6.
73. Dutt, N. D., Regazzoni, C. S., Rinner, B., & Yao, X. (2020). Self-awareness for autonomous systems. *Proceedings of the IEEE*, *108*(7), 971–975. https://doi.org/10.1109/JPROC.2020.2990784.
74. Economic, U. N., & Council, S. (1968). Vienna Convention on Road Traffic. http://www.unece.org/trans/conventn/crt1968e.pdf.
75. Edelkamp, S., Leue, S., & Lluch-Lafuente, A. (2004). Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools For Technology Transfer*, *5*(2–3), 247–267.
76. Emerson, E. A. (1990). Temporal and modal logic. In *Formal Models and Semantics* (pp. 995–1072). Elsevier, Newyork
77. Espejo-García, B., Martínez-Guanter, J., Pérez-Ruiz, M., López-Pellicer, F. J., & Zarazaga-Soria, F. J. (2018). Machine learning for automatic rule classification of agricultural regulations: A case study in Spain. *Computers and Electronics in Agriculture*, *150*, 343–352. https://doi.org/10.1016/j.compag.2018.05.007.
78. European Aviation Artificial Intelligence High Level Group (2020). The FLY AI report – demystifying and accelerating AI in aviation/ATM. URL https://www.eurocontrol.int/publication/fly-ai-report.

79. European Committee for Electrotechnical Standardisation. CENELEC web site. URL https://www.cenelec.eu/.
80. European Parliament (2016). Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679.
81. European Union Aviation Safety Agency. EASA web site. URL https://www.easa.europa.eu/.
82. FAA (November 2010). Qantas flight 32, airbus a380-842, vh-oqa. Online: https://lessonslearned.faa.gov/ll_main.cfm?TabID=1&LLID=83.
83. Falcone, Y., Krstic, S., Reger, G., & Traytel, D. (2018). A taxonomy for classifying runtime verification tools. In C. Colombo, & M. Leucker (Eds.), *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *Lecture Notes in Computer Science* (pp. 241–262). Springer. https://doi.org/10.1007/978-3-030-03769-7_14.
84. Farrell, M., Luckcuck, M., & Fisher, M. (2018). Robotics and Integrated Formal Methods: Necessity Meets Opportunity. In *Proceedings of 14th International Conference on Integrated Formal Methods (IFM'18)*, volume LNCS 11023 (pp. 161–171). Springer. https://doi.org/10.1007/978-3-319-98938-9_10.
85. Federal Aviation Administration. FAA web site. URL https://www.faa.gov/.
86. Federal Aviation Administration (2004). Title 14 code of Federal Regulations Part 145 approved training program – research and recommendations. URL https://www.faa.gov/about/initiatives/maintenance_hf/library/documents/media/human_factors_maintenance/ar04-36.pdf.
87. Federal Aviation Administration (2016). Part 107: Operation and certification of small unmanned aircraft systems. URL https://www.faa.gov/uas/media/RIN_2120-AJ60_Clean_Signed.pdf.
88. Ferrando, A., Ancona, D., & Mascardi, V. (2017). Decentralizing MAS monitoring with DecAMon. In *AAMAS* (pp. 239–248). ACM.
89. Ferrando, A., Dennis, L. A., Ancona, D., Fisher, M., & Mascardi, V. (2018). Recognising Assumption Violations in Autonomous Systems Verification. In *AAMAS* (pp. 1933–1935). International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.
90. Ferrando, A., Dennis, L. A., Ancona, D., Fisher, M., & Mascardi, V. (2018). Verifying and Validating Autonomous Systems: Towards an Integrated Approach. In *RV*, volume 11237 of *Lecture Notes in Computer Science* (pp. 263–281). Springer.
91. FINRA. Algorithmic trading: Rules. https://www.finra.org/rules-guidance/key-topics/algorithmic-trading#rules. Accessed 2019-10-15.
92. Fisher, M., Dennis, L. A., & Webster, M. P. (2013). Verifying autonomous systems. *Communications of the ACM*, *56*(9), 84–93. https://doi.org/10.1145/2494558.
93. Franchetti, F., Low, T. M., Mitsch, S., Mendoza, J. P., Gui, L., Phaosawasdi, A., et al. (2017). High-assurance spiral: End-to-end guarantees for robot and car control. *IEEE Control Systems Magazine*, *37*(2), 82–103.
94. Frauenberger, C., & Purgathofer, P. (2019). Ways of thinking in informatics. *Communications of the ACM*, *62*(7), 58–64. https://doi.org/10.1145/3329674.
95. FreeBSD. lint – a c program verifier. URL https://www.freebsd.org/cgi/man.cgi?query=lint&apropos=0&sektion=0&manpath=FreeBSD+11.1-RELEASE&arch=default&format=html.
96. Fulton, N., Ji, R., Platzer, A., et al. (2016). Proving autonomous vehicle and advanced driver assistance systems safety: final research report.
97. Galdino, A. L., Munoz, C., & Ayala-Rincón, M. (2007). Formal verification of an optimal air traffic conflict resolution and recovery algorithm. In *International Workshop on Logic, Language, Information, and Computation* (pp. 177–188). Springer, Berline , Heidelberg.
98. Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., & Rozier, K. Y. (July 2016). Model checking at scale: Automated air traffic control design space exploration. In *Proceedings of 28th International Conference on Computer Aided Verification (CAV 2016)*, volume 9780 of *LNCS* (pp. 3–22) Toronto, ON, Canada. Springer. https://doi.org/10.1007/978-3-319-41540-6_1.
99. Geist, J., Rozier, K. Y., & Schumann, J. (September 2014). Runtime observer pairs and bayesian network reasoners On-board FPGAs: Flight-certifiable system health management for embedded systems. In *Proceedings of the 14th International Conference on Runtime Verification (RV14)*, volume 8734 (pp. 215–230). Springer-Verlag.
100. Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., & Steiner, W. (2016). ARSENAL: automatic requirements specification extraction from natural language. In S. Rayadurgam, & O. Tkachuk (eds.), *NASA Formal Methods - 8th International Symposium, NFM 2016, Minneapolis,*

MN, USA, June 7-9, 2016, Proceedings, volume 9690 of *Lecture Notes in Computer Science* (pp. 41–46). Springer. https://doi.org/10.1007/978-3-319-40648-0_4.

101. Gunkel, D., & Bryson, J. J. (2014). Introduction to the special issue on machine morality: The machine as moral agent and patient. *Philosophy & Technology*, *27*(1), 5–8. https://doi.org/10.1007/s13347-014-0151-1.
102. Havelund, K., & Reger, G. (2017). Runtime verification logics a language design perspective. In *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, volume 10460 of *Lecture Notes in Computer Science* (pp. 310–338). Springer, New york
103. Heitmeyer, C. L. (2009). On the role of formal methods in software certification: An experience report. *Electronic Notes on Theoretical Computer Science*, *238*(4), 3–9. https://doi.org/10.1016/j.entcs.2009.09.001.
104. Heitmeyer, C. L., Archer, M., Leonard, E. I., & McLean, J. (2008). Applying formal methods to a certifiably secure software system. *IEEE Trans. Software Eng.*, *34*(1), 82–98. https://doi.org/10.1109/TSE.2007.70772.
105. Helle, P., Schamai, W., & Strobel, C. (2016). Testing of autonomous systems – challenges and current state-of-the-art. In *INCOSE International Symposium*, volume 26-1 (pp. 571–584). Wiley Online Library. https://doi.org/10.1002/j.2334-5837.2016.00179.x.
106. Henzinger, T. A., Ho, P.-H., & Wong-Toi, H. (1997). HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, *1*(1–2), 110–122.
107. Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Commun. ACM*, *12*(10), 576–580. https://doi.org/10.1145/363235.363259. ISSN 0001-0782.
108. Hodgkins, K. (5 Oct. 2011). Apple's Knowledge Navigator, Siri and the iPhone 4S. *Engadget*.
109. Holzmann, G. J. (2003). *The Spin Model Checker: Primer and Reference Manual*. : Addison-Wesley. ISBN 0-321-22862-6.
110. Huhns, M. N., & Singh, M. P. (1998). Agents on the web: Personal assistants. *IEEE Internet Computing*, *2*(5), 90–92.
111. Industry Research (2019). Software testing services market by product, end-users, and geography – global forecast and analysis 2019-2023. URL https://www.industryresearch.co/software-testing-services-market-by-product-end-users-and-geography-global-forecast-and-analysis-2019-2023-14620379.
112. Institute of Electrical and Electronics Engineers. The IEEE global initiative on ethics of autonomous and intelligent systems, a. URL https://standards.ieee.org/industry-connections/ec/autonomous-systems.html.
113. Institute of Electrical and Electronics Engineers. IEEE web site, b. URL https://www.ieee.org/.
114. Institute of Electrical and Electronics Engineers (2006). IEEE 1512-2006 – standard for common incident management message sets for use by emergency management centers. URL https://standards.ieee.org/standard/1512-2006.html.
115. Institute of Electrical and Electronics Engineers (2015). IEEE standard ontologies for robotics and automation. URL https://ieeexplore.ieee.org/document/7084073.
116. Institute of Electrical and Electronics Engineers (2016). P2020 – standard for automotive system image quality. URL https://standards.ieee.org/project/2020.html.
117. Institute of Electrical and Electronics Engineers (2016). P7000 – model process for addressing ethical concerns during system design. URL https://standards.ieee.org/project/7000.html.
118. Institute of Electrical and Electronics Engineers (2016c). P7001 – transparency of autonomous systems. URL https://standards.ieee.org/project/7001.html.
119. Institute of Electrical and Electronics Engineers (2016d). P7002 – data privacy process. URL https://standards.ieee.org/project/7002.html.
120. Institute of Electrical and Electronics Engineers (2017). P7003 – algorithmic bias considerations. URL https://standards.ieee.org/project/7003.html.
121. Institute of Electrical and Electronics Engineers (2017). P7006 – standard for personal data artificial intelligence (AI) agent. URL https://standards.ieee.org/project/7006.html.
122. Institute of Electrical and Electronics Engineers (2017c). P7007 – ontological standard for ethically driven robotics and automation systems. URL https://standards.ieee.org/project/7007.html.
123. Institute of Electrical and Electronics Engineers (2017d). P7008 – standard for ethically driven nudging for robotic, intelligent and autonomous systems. URL https://standards.ieee.org/project/7008.html.
124. Institute of Electrical and Electronics Engineers (2017e). P7009 – standard for fail-safe design of autonomous and semi-autonomous systems. URL https://standards.ieee.org/project/7009.html.

125. Institute of Electrical and Electronics Engineers – Robotics and Automation Society (2019). IEEE-RAS technical committee for verification of autonomous systems. URL https://www.ieee-ras.org/verification-of-autonomous-systems.
126. International Association of Public Transport – L'Union internationale des transports publics. UITP web site. URL https://www.uitp.org/.
127. International Civil Aviation Organization (2001). Annex 11 to the convention on international civil aviation, thirteenth edition. URL https://store.icao.int/products/annex-11-air-traffic-services.
128. International Electrotechnical Commission. IEC TC 107 – process management for avionics, a. URL https://www.iec.ch/dyn/www/f?p=103:7:0::::FSP_ORG_ID:1304.
129. International Electrotechnical Commission. IEC TC 97 – electrical installations for lighting and beaconing of aerodromes, b. URL https://www.iec.ch/dyn/www/f?p=103:7:0::::FSP_ORG_ID:1294.
130. International Electrotechnical Commission. IEC web site, c. URL https://www.iec.ch/.
131. International Electrotechnical Commission (2002). IEC 62278 – railway applications - specification and demonstration of reliability, availability, maintainability and safety (RAMS). URL https://webstore.iec.ch/publication/6747.
132. International Electrotechnical Commission (2010). Functional safety and IEC 61508. URL https://www.iec.ch/functionalsafety/.
133. International Electrotechnical Commission (2010). IEC 62278-3 – railway applications - specification and demonstration of reliability, availability, maintainability and safety (RAMS) - Part 3: Guide to the application of IEC 62278 for rolling stock RAM. URL https://webstore.iec.ch/publication/6746.
134. International Electrotechnical Commission (2016). IEC 62278-4 – railway applications - specification and demonstration of reliability, availability, maintainability and safety (RAMS) - Part 4: RAM risk and RAM life cycle aspects. URL https://webstore.iec.ch/publication/29621.
135. International Electrotechnical Commission (2017). IEC TC 69 – electric road vehicles and electric industrial trucks. URL https://www.iec.ch/dyn/www/f?p=103:7:0::::FSP_ORG_ID:1255.
136. International Electrotechnical Commission (2017). IEC TC 9 – electrical equipment and systems for railways. URL https://www.iec.ch/dyn/www/f?p=103:7:0::::FSP_ORG_ID,FSP_LANG_ID:1248,25.
137. International Electrotechnical Commission (2017c). IEC TR 60601-4-1 – medical electrical equipment – part 4-1: Guidance and interpretation - medical electrical equipment and medical electrical systems employing a degree of autonomy. URL https://webstore.iec.ch/publication/29312.
138. International Electrotechnical Commission (2019). IEC 63243 ED1 – interoperability and safety of dynamic wireless power transfer (WPT) for electric vehicles. URL https://www.iec.ch/dyn/www/f?p=103:38:1864379252239.
139. International Organization for Standardization. ISO web site. URL https://www.iso.org/.
140. International Organization for Standardization (1947). ISO/TC 20 – Aircraft and space vehicles. URL https://www.iso.org/committee/46484.html.
141. International Organization for Standardization (1951). ISO/TC 76 – transfusion, infusion and injection, and blood processing equipment for medical and pharmaceutical use. URL https://www.iso.org/committee/50044.html.
142. International Organization for Standardization (1988). ISO/TC 194 – biological and clinical evaluation of medical devices. URL https://www.iso.org/committee/54508.html.
143. International Organization for Standardization (1994). ISO/TC 210 – quality management and corresponding general aspects for medical devices. URL https://www.iso.org/committee/54892.html.
144. International Organization for Standardization (1998). ISO/TC 215 – health informatics. URL https://www.iso.org/committee/54960.html.
145. International Organization for Standardization (2012). ISO 21500 – guidance on project management. URL https://www.iso.org/standard/50003.html.
146. International Organization for Standardization (2012). ISO/TC 269 – railway applications. URL https://www.iso.org/committee/661629.html.
147. International Organization for Standardization (2014). ISO 13482 – robots and robotic devices – safety requirements for personal care robots. URL https://www.iso.org/standard/53820.html.
148. International Organization for Standardization (2015). ISO/TC 299 – robotics. URL https://www.iso.org/committee/5915511.html.
149. International Organization for Standardization (2016). ISO and road vehicles. URL https://www.iso.org/publication/PUB100292.html.
150. International Organization for Standardization (2018). ISO 21245 – railway applications – railway project planning process – guidance on railway project planning. URL https://www.iso.org/standard/74012.html.
151. International Organization for Standardization (2018). ISO 26262-1 – road vehicles – functional safety. URL https://www.iso.org/standard/68383.html.

152. International Organization for Standardization (2019). ISO and health. URL https://www.iso.org/publication/PUB100343.html.
153. International Organization for Standardization (ISO) (2014). ISO 13482 – robots and robotic devices — safety requirements for personal care robots. URL https://www.iso.org/standard/53820.html.
154. International Organization for Standardization (ISO) (2016). ISO/TS 15066 – robots and robotic devices – collaborative robots. URL https://www.iso.org/standard/62996.html.
155. International Organization for Standardization (ISO) (2017). ISO/TR 20218-2 – robotics – safety design for industrial robot systems – part 2: Manual load/unload stations. URL https://www.iso.org/standard/70584.html.
156. International Organization for Standardization (ISO) (2017). ISO/TR 23482-2 – robotics – application of ISO 13482 – part 2: Application guidelines. URL https://www.iso.org/standard/71627.html.
157. International Organization for Standardization (ISO) (2018). ISO/TR 20218-1 – robotics – safety design for industrial robot systems – part 1: End-effectors. URL https://www.iso.org/standard/69488.html.
158. International union of railways – Union Internationale des Chemins de fer. UIC web site. URL https://uic.org/.
159. Jasim, O. A., & Veres, S. M. (Oct 2017). Towards formal proofs of feedback control theory. In *Proc. 21st International Conference on System Theory, Control and Computing (ICSTCC)* (pp. 43–48). https://doi.org/10.1109/ICSTCC.2017.8107009.
160. Jennings, N. R., Sycara, K. P., & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, *1*(1), 7–38. https://doi.org/10.1023/A:1010090405266.
161. Jovinelly, J., & Netelkos, J. (2006). *The crafts and culture of a medieval guild*. New York, NY: Rosen Publishing.
162. Julius, A., & Pappas, G. (2009). Approximations of stochastic hybrid systems. *IEEE Transactions on Automatic Control*, *54*(6), 1193–1203.
163. Kempa, B., Zhang, P., Jones, P. H., Zambreno, J., & Rozier, K. Y. (September 2020). Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2. In *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume TBD of *Lecture Notes in Computer Science (LNCS)*, page TBD, Vienna, Austria: Springer. : TBD. URL http://research.temporallogic.org/papers/KZJZR20.pdf.
164. Kepuska, V., & Bohouta, G. (2018). Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 99–103). IEEE.
165. Khan, S. G., Herrmann, G., Pipe, A. G., Melhuish, C., & Spiers, A. (2010). Safe adaptive compliance control of a humanoid robotic arm with anti-windup compensation and posture Control. *Int. J. Social Robotics*, *2*(3), 305–319. https://doi.org/10.1007/s12369-010-0058-7.
166. Knight, J. C. (2002). Safety critical systems: challenges and directions. In W. Tracz, M. Young, & J. Magee (Eds.), *Proceedings of the 24th International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA* (pp. 547–550). ACM. https://doi.org/10.1145/581339.581406.
167. Kohlberg, L. (1969). Stage and sequence: The cognitive-developmental approach to socialization. In D. Goslin (Ed.), *Handbook of Socialization Theory and Research* (pp. 347–480). Rand McNally.
168. Kohlberg, L. (1981). *Essays on Moral Development. Volume I: The philosophy of moral development*. : Harper & Row.
169. Kohlberg, L. (1984). *Essays on Moral Development. Volume II: The psychology of moral development: the nature and validity of moral stages*. : Harper & Row.
170. Kong, J., & Lomuscio, A. (2017). Symbolic model checking multi-agent systems against CTL*K specifications. In K. Larson, M. Winikoff, S. Das, & E. H. Durfee (Eds.), *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017* (pp. 114–122). ACM. URL http://dl.acm.org/citation.cfm?id=3091147.
171. Kong, J., & Lomuscio, A. (2018). Model checking multi-agent systems against LDLK specifications on finite traces. In E. André, S. Koenig, M. Dastani, & G. Sukthankar (Eds.), *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018* (pp. 166–174). International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM. URL http://dl.acm.org/citation.cfm?id=3237414.
172. Kowalski, R. A., & Sadri, F. (1996). Towards a unified agent architecture that combines rationality with reactivity. In D. Pedreschi, & C. Zaniolo (Eds.), *Logic in Databases, International Workshop*

LID'96, San Miniato, Italy, July 1-2, 1996, Proceedings, volume 1154 of *Lecture Notes in Computer Science* (pp. 137–149). Springer. https://doi.org/10.1007/BFb0031739.

173. Leveson, N. G., & Turner, C. S. (1993). An investigation of the Therac-25 accidents. *Computer*, *26*(7), 18–41. https://doi.org/10.1109/MC.1993.274940. ISSN 1558-0814.
174. Levine, D. M. (23May 2013). A day in the quiet life of a NYSE floor trader. *Fortune*. URL https://fortune.com/2013/05/29/a-day-in-the-quiet-life-of-a-nyse-floor-trader/.
175. Li, J., & Rozier, K. Y. (November 2018). MLTL Benchmark Generation via Formula Progression. In *Proceedings of the 18th International Conference on Runtime Verification (RV18)*, Limassol, Cyprus. Springer-Verlag.
176. Li, J., Vardi, M., & Rozier, K. Y. (July 2019). Satisfiability checking for mission-time LTL. In *Proceedings of 31st International Conference on Computer Aided Verification (CAV'19)*, LNCS. Springer. https://doi.org/10.1007/978-3-030-25543-5_1.
177. Lomuscio, A., & Raimondi, F. (2006). Model checking knowledge, strategies, and games in multi-agent systems. In H. Nakashima, M. P. Wellman, G. Weiss, & P. Stone (Eds.), *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006* (pp. 161–168). ACM. https://doi.org/10.1145/1160633.1160660.
178. Luckcuck, M., Farrell, M., Dennis, L., Dixon, C., & Fisher, M. (2019). Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys*, *52*(5), 100:1–100:41. https://doi.org/10.1145/3342355.
179. Luckow, K. S., & Păsăreanu, C. S. (2014). Symbolic pathfinder v7. *ACM SIGSOFT Software Engineering Notes*, *39*(1), 1–5.
180. Maggi, F. M., Montali, M., Westergaard, M., & van der Aalst, W. M. P. (2011). Monitoring business constraints with linear temporal logic: An approach based on colored automata. In S. Rinderle-Ma, F. Toumani, & K. Wolf (Eds.), *Proceedings of 9th International Conference on Business Process Management (BPM'11)*, volume 6896 of *LNCS* (pp. 132–147). Springer. https://doi.org/10.1007/978-3-642-23059-2_13.
181. Marr, B. (2017). The biggest challenges facing artificial intelligence (AI) in business and society. *Forbes*. URL https://www.forbes.com/sites/bernardmarr/2017/07/13/the-biggest-challenges-facing-artificial-intelligence-ai-in-business-and-society/.
182. MathWorks. Polyspace bug finder. URL https://in.mathworks.com/products/polyspace-bug-finder.html.
183. Mattarei, C., Cimatti, A., Gario, M., Tonetta, S., Rozier, K. Y., & (September 2015). Comparing different functional allocations in automated air traffic control design. In *Proceedings of Formal Methods in Computer-Aided Design (FMCAD,. (2015)*. Austin, Texas, U.S.A, IEEE/ACM.
184. Matthias, A. (2011). Robot lies in health care: when is deception morally permissible? *Kennedy Institute of Ethics Journal*, *25*(2), 279–301.
185. McMillan, K. L. (1999). The SMV language. *Cadence Berkeley Labs* (pp. 1–49).
186. Merriam-Webster Dictionary (2020). Definition of 'reliable'. URL https://www.merriam-webster.com/dictionary/reliable.
187. Moosbrugger, P., Rozier, K. Y., & Schumann, J. (April 2017). R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems. In *Formal Methods in System Design (FMSD)* (pp. 1–31). Springer-Verlag. https://doi.org/10.1007/s10703-017-0275-x.
188. Munoz, C., Narkawicz, A., & Chamberlain, J. (2013). A TCAS-II resolution advisory detection algorithm. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4622.
189. Muñoz, C., Narkawicz, A., Hagen, G., Upchurch, J., Dutle, A., Consiglio, M., & Chamberlain, J. (2015). Daidalus: detect and avoid alerting logic for unmanned systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)* (pp. 5A1–1). IEEE.
190. Musuvathi, M., Engler, D. R., et al. (2004). Model checking large network protocol implementations. *NSDI*, *4*, 12–12.
191. Negroponte, N. (1996). *Being Digital*. New York, NY, USA: Random House. ISBN 0-679-43919-6.
192. Nguyen, C. D., Perini, A., Bernon, C., Pavón, J., & Thangarajah, J. (2009). Testing in multi-agent systems. In M. P. Gleizes, & J. J. Gómez-Sanz (Eds.), Agent-Oriented Software Engineering X - 10th International Workshop, AOSE. (2009). *Budapest, Hungary, May 11–12, 2009, Revised Selected Papers* (Vol. 6038, pp. 180–190)., Lecture Notes in Computer Science New york: Springer.
193. Patchett, C., Jump, M., & Fisher, M. (2015). Institution of engineering and technology:in engineering and technology reference. *Safety and Certification of Unmanned Air Systems*,. https://doi.org/10.1049/etr.2015.0009.

194. Paulson, L. C. (1994). *A Generic Theorem Prover* (Vol. 828)., Lecture Notes in Computer Science New york: Springer.
195. Penczek, W., & Lomuscio, A. (2003). Verifying epistemic properties of multi-agent systems via bounded model checking. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings* (pp. 209–216). ACM. https://doi.org/10.1145/860575.860609.
196. Perez, I., Dedden, F., & Goodloe, A. (2020). Copilot 3. Technical Report NASA/TM-2020-220587, National Aeronautics and Space Administration.
197. Pietrantuono, R., & Russo, S. (2018). Robotics software engineering and certification: Issues and challenges. In S. Ghosh, R. Natella, B. Cukic, R. Poston, & N. Laranjeiro (Eds.), *2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Memphis, TN, USA, October 15-18, 2018* (pp. 308–312). IEEE Computer Society. https://doi.org/10.1109/ISSREW.2018.00023.
198. Pietronudo, E. (2018). ”Japanese women's language” and artificial intelligence: Azuma Hikari, gender stereotypes and gender norms. http://hdl.handle.net/10579/12791.
199. Pike, L. (2007). Modeling time-triggered protocols and verifying their real-time schedules. In *Formal Methods in Computer Aided Design (FMCAD'07)* (pp. 231–238). IEEE.
200. platoon. Current State of EU Legislation- Cooperative Dynamic Formation of Platoons for Safe and Energy-optimized Goods Transportation. URL http://www.companion-project.eu/wp-content/uploads/COMPANION-D2.2-Current-state-of-the-EU-legislation.pdf.
201. Platzer, A. (2010). *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Heidelberg: Springer. https://doi.org/10.1007/978-3-642-14509-4. ISBN 978-3-642-14508-7.
202. Platzer, A., & Quesel, J.-D. (2008). KeyMaera: A Hybrid Theorem Prover for Hybrid Systems. In A. Armando, P. Baumgartner, & G. Dowek (Eds.), *Proceedings of 4th International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *LNCS* (pp. 171–178). Springer.
203. Poore, J. H., Mills, H. D., & Mutchler, D. (1993). Planning and certifying software system reliability. *IEEE Software*, *10*(1), 88–99.
204. Quirchmayr, T. (2018). *Retrospective semi-automated software feature extraction from natural language user manuals*. PhD thesis, University of Heidelberg, Germany. URL http://www.ub.uni-heidelberg.de/archiv/25322.
205. Radio Technical Commission for Aeronautics. RTCA web site. URL https://www.rtca.org/.
206. Radio Technical Commission for Aeronautics (1992). DO-178B – software considerations in airborne systems and equipment certification. URL https://www.rtca.org/content/standards-guidance-materials.
207. Radio Technical Commission for Aeronautics (1992). DO-278A – software integrity assurance considerations for communication, navigation, surveillance and air traffic management (CNS/ATM) systems. URL https://www.rtca.org/content/standards-guidance-materials.
208. Radio Technical Commission for Aeronautics (2000). DO-254 – design assurance guidance for airborne electronic hardware. URL https://www.rtca.org/content/standards-guidance-materials.
209. Radio Technical Commission for Aeronautics (2011). DO-333 – formal methods supplement to DO-178C and DO-278A. URL https://www.rtca.org/content/standards-guidance-materials.
210. Radio Technical Commission for Aeronautics (2012). DO-178C/ED-12C – software considerations in airborne systems and equipment certification. URL https://www.rtca.org/content/standards-guidance-materials.
211. Raman, V., Lignos, C., Finucane, C., Lee, K. C., Marcus, M. P., & Kress-Gazit, H. (2013). Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language. volume 2. Citeseer.
212. Ramesh, B., & Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, *27*(1), 58–93. https://doi.org/10.1109/32.895989.
213. Reinbacher, T., Rozier, K. Y., & Schumann, J. (2014). Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proceedings of 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume LNCS 8413 (pp. 357–372). Springer. https://doi.org/10.1007/978-3-642-54862-8_24.
214. Rinehart, D. J., Knight, J. C., & Rowanhill, J. (2017). Understanding what it means for assurance cases to "work". Technical report, NASA. NASA/CR–2017-219582.
215. Rosu, G. (2012). On safety properties and their monitoring. *Science Annals of Computer Science.*, *22*(2), 327–365.
216. Rozier, K., & Vardi, M. (2010). LTL satisfiability checking. *International Journal on Software Tools for Technology Transfer (STTT)*, *12*(2), 123–137. https://doi.org/10.1007/s10009-010-0140-3.

217. Rozier, K. Y. (2011). Linear Temporal Logic Symbolic Model Checking. *Computer Science Review Journal*, *5*(2), 163–203.
218. Rozier, K. Y. (2016). Specification: The biggest bottleneck in formal methods and autonomy. In *Proceedings of 8th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE'16)*, volume LNCS 9971 (pp. 1–19). Springer. https://doi.org/10.1007/978-3-319-48869-1_2.
219. Rozier, K. Y. (April 2019). From simulation to runtime verification and back: Connecting single-run verification techniques. In *Proceedings of the Spring Simulation Conference (SpringSim)* (pp. 1–10), Tucson, AZ, USA. Society for Modeling & Simulation International. https://dl.acm.org/doi/10.5555/3338027.3338054.
220. Rozier, K. Y., & Schumann, J. (2017). R2U2: Tool overview. In *Proceedings of International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES)* (pp. 138–156). https://doi.org/10.29007/5pch.
221. SAE International (2018). SAE J3016_201806 – taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. URL https://www.sae.org/standards/content/j3016_201806/.
222. Salem, M., Lakatos, G., Amirabdollahian, F., & Dautenhahn, K. (2015). Would you trust a (faulty) robot?: Effects of error, task type and personality on human-robot cooperation and trust. In *Proceedings of 10th ACM/IEEE International Conference on Human-Robot Interaction, HRI 2015, Portland, OR, USA, March 2-5, 2015*, (pp. 141–148). ACM.
223. Salem, M., Lakatos, G., Amirabdollahian, F., & Dautenhahn, K. (2015). Towards Safe and Trustworthy Social Robots: Ethical Challenges and Practical Issues. In *Proc. 7th International Conference on Social Robotics (ICSR)*, volume 9388 of *LNCS* (pp. 584–593). Springer.
224. Sartre. SARTRE project. URL https://cordis.europa.eu/project/rcn/92577/brief/en.
225. Schlatow, J., Möstl, M., Ernst, R., Nolte, M., Jatzkowski, I., Maurer, M., Herber, C., & Herkersdorf, A. (2017). Self-awareness in autonomous automotive systems. In D. Atienza, & G. D. Natale (Eds.), *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017* (pp. 1050–1055). IEEE. https://doi.org/10.23919/DATE.2017.7927145.
226. Schumann, J., Moosbrugger, P., & Rozier, K. Y. (September 2016). Runtime Analysis with R2U2: A Tool Exhibition Report. In *Proceedings of the 16th International Conference on Runtime Verification (RV16)*. Madrid, Spain: Springer-Verlag.
227. Scrapper, C., Balakirsky, S., & Messina, E. (2006). MOAST and USARSim: a combined framework for the development and testing of autonomous systems. In *Unmanned Systems Technology VIII*, volume 6230, page 62301T. International Society for Optics and Photonics.
228. SCSC - The Safety-Critical Systems Club. SCSC – goal structuring notation community standard (version 2). URL https://scsc.uk/scsc-141B.
229. Shankar, N. (2008). Trust and automation in verification tools. In *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings* (pp. 4–17). https://doi.org/10.1007/978-3-540-88387-6_3.
230. Sharkey, A., & Wood, N. (2014). The paro seal robot: demeaning or enabling. In *Proceedings of AISB*, volume 36.
231. Stout, D. (2011). Stone toolmaking and the evolution of human culture and cognition. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *366*(1567), 1050–1059.
232. Swaroop, D. (1997). String stability of interconnected systems: An application to platooning in automated highway systems. *California Partners for Advanced Transit and Highways (PATH).*
233. Tabakov, D., Rozier, K. Y., & Vardi, M. Y. (January 2012). Optimized temporal monitors for SystemC. *Formal Methods in System Design*, *41*(3), 236–268. https://doi.org/10.1007/s10703-011-0139-8.
234. The European Parliament (2018). Regulation (EU) 2018/1139 of the European Parliament and of the Council of 4 July 2018 on common rules in the field of civil aviation and establishing a European Union Aviation Safety Agency, and amending Regulations (EC) No 2111/2005, (EC) No 1008/2008, (EU) No 996/2010, (EU) No 376/2014 and Directives 2014/30/EU and 2014/53/EU of the European Parliament and of the Council, and repealing Regulations (EC) No 552/2004 and (EC) No 216/2008 of the European Parliament and of the Council and Council Regulation (EEC) No 3922/91 (Text with EEA relevance). URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32018R1139.
235. The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems, (Ed.) (2019). *Ethically Aligned Design: A Vision for Prioritizing Human Well-being with Autonomous and Intelligent Systems*. IEEE. URL https://standards.ieee.org/content/ieee-standards/en/industry-connections/ec/ autonomous-systems.html.

236. The Software Testing Help (STH) Blog (2019). Top 40 static code analysis tools (best source code analysis tools). URL https://www.softwaretestinghelp.com/tools/top-40-static-code-analysis-tools/.

237. Tolmeijer, S., Weiss, A., Hanheide, M., Lindner, F., Powers, T. M., Dixon, C., & Tielman, M. L. (2020). Taxonomy of trust-relevant failures and mitigation strategies. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction (HRI)* (pp. 3-12). Association for Computing Machinery. https://doi.org/10.1145/3319502.3374793.

238. Tomayko, J. E. (2003). The story of self-repairing flight control systems. In C. Gelzer (Ed.), *Dryden Historical Study No. 1.* : NASA Dryden Flight Research Center.

239. Torens, C., Adolf, F., & Goormann, L. (2014). Certification and software verification considerations for autonomous unmanned aircraft. *Journal of Aerospace Information System.*, *11*(10), 649–664. https://doi.org/10.2514/1.I010163.

240. Tuncali, C. E., Fainekos, G., Prokhorov, D., Ito, H., & Kapinski, J. (2019). Requirements-driven test generation for autonomous vehicles with machine learning components. *IEEE Transactions on Intelligent Vehicles*, *5*(2), 265–280.

241. U.S. Department of Transportation (2016). Federal automated vehicles policy. URL https://www.transportation.gov/AV/federal-automated-vehicles-policy-september-2016.

242. van der Aalst, W. M. P. (2002). Making work flow: On the application of petri nets to business process management. In J. Esparza, & C. Lakos (Eds.), *Proceedings of 23rd International Conference on Applications and Theory of Petri Nets (ICATPN'02)*, volume 2360 of *Lecture Notes in Computer Science* (pp. 1–22). Springer. https://doi.org/10.1007/3-540-48068-4_1.

243. van der Aalst, W. M. P. (2011). *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. New york: Springer. https://doi.org/10.1007/978-3-642-19345-3. ISBN 978-3-642-19344-6.

244. Visser, W., Havelund, K., Brat, G. P., Park, S., & Lerda, F. (2003). Model checking programs. *Automated Software Engineering*, *10*(2), 203–232.

245. Wada, K., Shibata, T., Asada, T., & Musha, T. (2007). Robot therapy for prevention of dementia at home. *Journal of Robotics and Mechatronics*, *19*(6), 691.

246. Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K., et al. (2016). Toward reliable autonomous robotic assistants through formal verification: A Case Study. *IEEE Transactions on Human-Machine Systems*, *46*(2), 186–196. https://doi.org/10.1109/THMS.2015.2425139. ISSN 2168-2291.

247. Webster, M. P., Cameron, N., Fisher, M., & Jump, M. (2014). Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. *Journal of Aerospace Information System.*, *11*(5), 258–279. https://doi.org/10.2514/1.I010096.

248. Whitehurst, R. A., & Lunt, T. F. (1989). The sea view verification. In *Proceedings of 2nd IEEE Computer Security Foundations Workshop (CSFW'89)* (pp. 125–132). IEEE Computer Society. https://doi.org/10.1109/CSFW.1989.40595.

249. Winfield, A. F. T., Michael, K., Pitt, J., & Evers, V. (2019). Machine ethics: The design and governance of ethical AI and autonomous systems. *Proceedings of the IEEE*, *107*(3), 509–517. https://doi.org/10.1109/JPROC.2019.2900622.

250. Winikoff, M. (2017). BDI agent testability revisited. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, *31*(5), 1094–1132. https://doi.org/10.1007/s10458-016-9356-2.

251. Winikoff, M., & Cranefield, S. (2014). On the testability of BDI agent systems. *Journal of Artificial Intelligence Research*, *51*, 71–131. https://doi.org/10.1613/jair.4458.

252. Wohlin, C., & Runeson, P. (1994). Certification of software components. *IEEE Trans. Software Eng.*, *20*(6), 494–499.

253. Woodman, R., Winfield, A. F. T., Harper, C. J., & Fraser, M. (2012). Building safer robots: Safety driven control. *International Journal of Robotics Research*, *31*(13), 1603–1626. https://doi.org/10.1177/0278364912459665.

254. Wooldridge, M., & Jennings, N. R. (Eds.). (1995). *Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 8–9, 1994, Proceedings*, volume LNCS 890. Springer. https://doi.org/10.1007/3-540-58855-8.

255. Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: theory and practice. *Knowledge Eng. Review*, *10*(2), 115–152. https://doi.org/10.1017/S0269888900008122.

256. Working Party on Automated/autonomous and Connected Vehicles, Economic Commission for Europe (2020). Proposal for a new UN regulation on uniform provisions concerning the approval of vehicles with regards to Automated Lane Keeping System. URL https://undocs.org/ECE/TRANS/WP.29/2020/81.

257. Xiao, L., Lewis, P. H., & Dasmahapatra, S. (2008). Secure Interaction Models for the HealthAgents System. In *Proc. 27th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, volume 5219 of *LNCS* (pp. 167–180). Springer. ISBN 978-3-540-87697-7.
258. Yang, M., & Chow, K.-P. (2015). An information extraction framework for digital forensic investigations. In *IFIP International Conference on Digital Forensics*, (pp. 61–76). Springer.
259. Yorke-Smith, N., Saadati, S., Myers, K. L., & Morley, D. N. (2009). Like an intuitive and courteous butler: A proactive personal agent for task management. In *Proceedings of 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'09)* (pp. 337–344).
260. Yu, H., Lin, C.-W., & Kim, B. (2016). Automotive software certification: current status and challenges. *SAE International journal of passenger cars-electronic and electrical systems*, *9*, 74–80.
261. Zhang, N., Wang, J., & Ma, Y. (2018). Mining domain knowledge on service goals from textual service descriptions. *IEEE Transactions on Services Computing* (pp. 1–1). ISSN 1939-1374. https://doi.org/10.1109/TSC.2017.2693147.
262. Zhao, Y., & Rozier, K. Y. (2014). Formal specification and verification of a coordination protocol for an automated air traffic control system. *Science of Computer Programming Journal*, *96*(3), 337–353.
263. Zhao, Y., & Rozier, K. Y. (November 2014). Probabilistic model checking for comparative analysis of automated air traffic control systems. In *Proceedings of the 33rd IEEE/ACM International Conference On Computer-Aided Design (ICCAD 2014)* (pp. 690–695). San Jose, California, U.S.A.. IEEE/ACM.