Safe Artificial Intelligence and Formal Methods
(position paper)

Emil Vassev

Abstract In one aspect of our life or another, today we all live with AI. For example, the mechanisms behind the search engines operating on the Internet do not just retrieve information, but also constantly learn how to respond more rapidly and usefully to our requests. Although framed by its human inventors, this AI is getting stronger and more powerful every day to go beyond the original human intentions in the future. One of the major questions emerging along with the propagation of AI in both technology and life is about safety in AI. This paper presents the author's view about how formal methods can assist us in building safer and reliable AI.

1 Introduction

AI depends on our ability to efficiently transfer knowledge to software-intensive systems. A computerized machine can be considered as one exhibiting AI when it has the basic capabilities to transfer data into context-relevant information and then that information into conclusions exhibiting knowledge. Going further, we can say that AI is only possible in the presence of artificial awareness [12], one by which we can transfer knowledge to machines. Artificial awareness entails much more than computerized knowledge, however. It must also incorporate means by which a computerized machine can perceive events and gather data about its external and internal worlds. Therefore, to exhibit awareness, intelligent systems must sense and analyze components as well as the environment in which they operate. Determining the state of each component and its status relative to performance standards, or service-level objectives, is therefore vital for an aware system. Such systems should be able to notice changes, understand their implications, and apply both pattern analysis and pattern recognition to determine normal and abnormal states. In other

Emil Vassev
Lero–the Irish Software Research Centre, University of Limerick, Limerick, Ireland e-mail: emil.vassev@lero.ie

words, awareness is conceptually a product of representing, processing, and monitoring knowledge. Therefore, AI requires knowledge representation, which can be considered as a formal specification of the "brain" of an AI system. Moreover, to allow for learning, we must consider an open-world model of this "machine brain".

2 Artificial Intelligence and Safety

But, how to build safe AI systems? With regard to system safety, there seem to be at least two "cultures" among the AI scientists. One culture emphasizes the limitations of systems that are amenable to formal methods (e.g., machine learning techniques), and advises that developers use traditional software development methods to build a functional system, and try to make it safe near the end of the process. The other culture mainly involves people working on safety-critical systems and it tends to think that getting strong safety guarantees is generally only possible when a system is designed "from the ground up" with safety in mind.

I believe, both research "cultures" have their niche within AI. Both cultures lean towards the use of open-world modeling of the AI by using formal methods. The difference lies mainly in the importance of the safety requirements, which justifies both approaches. Note that AI is a sort of superior control mechanism that exclusively relies on the functionality of the system to both detect safety hazards and pursue safety procedures. Therefore, in all cases AI is limited by system functionality and systems designed "from the ground up with safety in mind" are presumably designed with explicit safety-related functionality, and thus, their AI is less limited when it comes to safety.

For many NASA and ESA systems [17], safety is an especially important source of requirements. Requirements engineers can express safety requirements as a set of features and procedures that ensure predictable system performance under normal and abnormal conditions. Furthermore, AI engineers might rely on safety requirements to derive special self-* objectives controlling the consequences of unplanned events or accidents [13, 14]. You can think about the self-* objectives as AI objectives driving the system in critical situations employing self-adaptive behavior. Safety standards might be a good source of safety requirements and consecutively on safety-related self-* objectives. Such self-* objectives may provide for fault-tolerance behavior, bounding failure probability, and adhering to proven practices and standards. Explicit safety requirements provide a key way to maintain safety-related knowledge within a machine brain of what is important for safety. In typical practice, safety-related AI requirements can be derived by a four-stage process [14]:

1. Hazard identification – all the hazards exhibited by the system are identified. A hazard might be regarded as a condition - situation, event, etc., that may lead to an accident.
2. Hazard analysis – possible causes of the system's hazards are explored and recorded. Essentially, this step identifies all processes, combinations of events, and sequences that can lead from a "normal" or "safe" state to an accident.

Success in this step means that we now understand how the system can get to an accident.
3. Identifying safety capabilities – a key step is to identify the capabilities (functionality) the system needs to have in order to perform its goals and remain safe. It is very likely that some of the capabilities have been already identified by for the purpose of other self-* objectives.
4. Requirements derivation – once the set of hazards is known, and their causation is understood, engineers can derive safety requirements that either prevent the hazards occurring or mitigate the resulting accidents via self-* objectives.

3 AI and Technological Singularity

But what will happen when the AI is programmed to *self-adapt* its hazard identification capabilities to improve the same or to identify new hazards that are not originally planned to be tackled. Well, we will most probably get to the next level of AI where it evolves and goes beyond its original meaning. This situation can be addressed as *technological singularity* [18]. Note that the term *singularity* has been used in math to describe an asymptote-like situation where normal rules no longer apply. For example, an originally programmed AI can stop detecting specific hazards, just because its evaluation criteria has evolved, and these hazards are not hazards anymore. From this point forward, we will be not tat far from the moment in the future when our technology's intelligence exceeds our own.

Such AI will be both powerful and dangerous. Why dangerous? The answer lays in the eventual damage - direct or indirect, that can be caused by overlooked hazards or misinterpreted human intentions. For example, your email spam filter can be loaded with intelligence about how to figure out what is spam and what is not and it will start to learn and tailor its intelligence to you as it gets experience with your particular preferences. However, you often delete emails that you want to read but do not want to keep in your email box. This can be misinterpreted by the spam filter's AI and it can start filtering these important messages for you.

4 No System Can Be 100% Safe

Generally speaking, formal methods strive to build software right (and thus, reliable) by eliminating flaws, e.g., requirements flaws. Formal method tools allow comprehensive analysis of requirements and design and eventually near-to-complete exploration of system behavior, including fault conditions. However, good requirements formalization depends mainly on the analytical skills of the requirements engineers along with the proper use of the formal methods in hand. Hence, errors can be introduced when capturing or implementing safety requirements. This is maybe the main reason why, although efficient in terms of capacity of the dedicated analysis

tools such as theorem provers and model checkers, formal methods actually do not eliminate the need of testing.

In regards with safety requirements, the application of formal methods can only add on safety. Even if we assume that proper testing can capture all the safety flaws that we may capture with formal verification, with proper use of formal methods we can always improve the quality of requirements and eventually derive more efficient test cases. Moreover, formal methods can be used to create formal specifications, which subsequently can be used for automatic test case generation. Hence, in exchange for the extra work put to formally specify the safety requirements of a system, you get not only the possibility to formally verify and validate these requirements, but also to more efficiently test their implementation.

It is evident that 100% safety cannot be guaranteed, but when properly used, formal methods can significantly contribute to safety by not replacing, but complementing testing. The quantitative measure of how much safety can be gained with formal methods may be regarded in three aspects:

1. Formal verification and validation allows for early detection of safety flaws, i.e., before implementation.
2. High quality of safety requirements improves the design and implementation of these requirements.
3. Formally specified safety requirements assist in the derivation and generation of efficient test cases.

To be more specific, although it really depends on the complexity of the system in question, my intuition is that these three aspects complement each other and together they may help us build a system with up to 99% safety guarantee. This principle can be eventually applied to improve safety in AI by emphasizing its ability to *autonomously* tackle various hazards. Of course, this excludes the AI that is elevated to the *technological singularity* level (see Section 3). For such AI, some form of formal validation of "desired" technological singularity will help with the safety guarantee. Eventually, some sort of analysis and formal framing of the system's artificial awareness can help with the validation of "desired" technological singularity.

5 What can be Formalized?

Contemporary formal verification techniques (e.g., model checking [2]) rely on state-transition models where objects or entities are specified with states they can be in and associated with functions that are performed to change states or object characteristics. Therefore, basically every system property that can be measured or quantified, or qualified as a function can be formalized for the needs of formal verification. Usually, the traditional types of requirements – functional and non-functional (e.g., data requirements, quality requirements, time constraints, etc.), are used to provide a specific description of functions and characteristics that address

the general purpose of the system. The formal verification techniques use the formal specification of such requirements to check desired safety and liveness properties. For example, to specify safety properties of a system, we need to formalize "nothing bad will happen to the system", which can be done via the formalization of non-desirable system states along with the formalization of behavior that will never lead the system to these states.

Obviously, the formalization of well-defined properties (e.g., with proper states expressed via boundaries, data range, outputs, etc.) is a straightforward task [19]. However, it is not that easy to formalize uncertainty, e.g., liveness properties (something good will eventually happen). Although, probabilistic theories such as the classical and quantum theories, help us formalize "degrees of truth" and deal with approximate conclusions rather with exact ones, the verification tools for fuzzy control systems are not efficient due to the huge state-explosion problem [2]. Moreover, testing such systems is not efficient as well, simply because, statistical evidence for their correct behavior may be not enough. Hence, any property that requires a progressive evaluation (or partial satisfaction, e.g., soft goals) is difficult and often impossible to be formalized for use in formally verified systems.

Other properties that are "intuitively desirable" (especially by AI) but still cannot be formalized today are human behavior and principles, related to cultural differences, ethics, feelings, etc. The problem is that with the formal approaches today we cannot express, for example, emotional bias as a meaningful system state.

6 Safe Self-driving Car Example

The example presented here should be regarded with the insight that "100% safety is not possible", especially when the system in question (e.g., a self-driving car) engages in interaction with a non-deterministic and open-world environment [19] (see Figure 1). What we should do though, to maximize the safety guarantee that "the car would never injure a pedestrian" is to determine all the critical situations involving the car itself in close proximity to pedestrians. Then we shall formalize these situations as system and environment states and formalize self-adaptive behavior (e.g., as self-* objectives [13, 14]) driving the car in such situations [14, 15]. For example, a situation could be defined as "all the car's systems are in operational condition and the car is passing by a school". To increase safety in this situation, we may formalize a self-adaptive behavior such as "automatically decrease the speed down to 20 mph when getting in close proximity to children or a school".

Further, we need to specify situations involving close proximity to pedestrians (e.g., crossing pedestrians) and car states emphasizing damages or malfunction of the driving system, e.g., flat tires, malfunctioning steering wheel, malfunctioning brakes, etc. For example, we may specify a self-adaptive behavior "automatically turn off the engine when the brake system is malfunctioning and the car is getting in close proximity to pedestrians".

Fig. 1 Self-driving Car Interacts with the Environment [7]

Other important situations should involve severe weather conditions introducing hazards on the road, e.g., snow storm, ice, low visibility (formalized as environment states), and the car getting in close proximity to pedestrians. In such situations, formalized self-adaptive behavior should automatically enforce low speed, turning lights on, turning wipers on, etc.

In this example, the self-* objectives shall be driven by an AI reasoner, so different situations will be recognized and handled by an appropriate behavior.

7 Deductive Guarantees and Probabilistic Guarantees

Many of the deductive proofs for safety properties in today's formally verified systems are already "probabilistic" in the sense that the designers have some subjective uncertainty as to whether the formal specification accurately captures the intuitively desirable safety properties, and (less likely) whether there was an error in the proof somewhere.

With deductive guarantees [11] a formal verification actually provides true statements that demonstrate that desired safety properties are held. Such a verification process is deterministic and a complete proof is required to guarantee the correctness of safety properties. For example, such a proof can be equipped with deterministic rules and expressed in the classical first-order logic (or in high-order logic if we use Isabelle to run a deductive verification). On the other hand, with the probabilistic guarantees we can accept that a complete proof is not necessary and safety properties can be verified with some degree of uncertainty. Basically, the probabilistic guarantees can be regarded as a result of quantification of uncertainty in both the verification parameters and subsequent predictions. With the Bayesian methods [3], for example, we quantify our uncertainty as prior distribution of our beliefs we have in the values of certain properties. Moreover, we also embed likelihood in the properties formalization, i.e., how likely is it that we would observe a certain value in particular conditions. You may think about it as the likelihood of holding certain

safety properties in specific situations. Then, the probabilistic guarantees assert in a natural way "likely" properties over the possibilities that we envision.

Unfortunately, deductive guarantees can be provided only for simple safety properties, because their complete proof often unavoidably does not terminate. Although deductive verification may deal with infinite state systems, its automation is limited, which is mainly due to the decidability of the logical reasoning (first-order logic and its extensions such as high-order logic are not decidable, or they are rather semi-decidable [8]). If we go back to our example with the self-driving car (see Section 6), we may supply all the needed deterministic rules expressing our safety requirements (e.g., speed limit of 20 mph when passing by a school), but the complete proof eventually cannot be achieved, because although the desired conclusion follows from some of the premises, other premises may eventually lead to resolution refutation. That's it, two sets of premises may lead to different proof results.

The probabilistic guarantees [2] are not as complete as the deductive ones, but they may deal with more complex properties, e.g., where a larger number of states can be required. Of course, this tradeoff should be considered when evaluating the results of any probabilistic formal verification. So, if we ask ourselves how much confidence in system's safety is gained with formal methods, probabilistic guarantees bring less confidence than deductive ones, but they may bring some extra confidence to safety properties that cannot be handled otherwise.

It is important to mention that abstraction [4] is the most efficient solution to the state-explosion problem (and respectively, to the problem of deductive guarantees decidability). With abstraction the size of the state space is reduced by aggregating state transitions into coarser-grained state transitions. The technique effectively reduces the total amount of states to be considered but is likely to reduce the granularity of the system to a point where it no longer adequately represents that system. The problem is that although the abstract model (e.g, the formalization of safety properties) is relatively small it should also be precise enough to adequately represent the original system.

Therefore, in order to obtain better results, we shall consider both verification approaches and eventually apply these together. For example, we may formalize with the presumption that both deductive and probabilistic guarantees can be obtained in a sort of compositional verification where we may apply both approaches to different safety properties, and eventually combine the results under the characteristics of global safety invariants. Such invariants can be classified as: goal invariants, behavior invariants, interaction invariants and resource invariants [10].

8 Improving our Current Verification Toolset

Maybe the most popular technique for formal verification is model checking [2] where the properties are expressed in a temporal logic and the system formalization is turned into a state machine. The model checking methods verify if the desired properties hold in all the reachable states of a system, which is basically a proof

that properties will hold during the execution of that system. State explosion is the main issue model checking is facing today. This problem is getting even bigger when it comes to concurrent systems where the number of states is exponential to the number of concurrent processes. So, basically, model checking is an efficient and powerful verification method, but only when applied to finite, yet small state spaces.

Here, to improve the current verification toolset, on the one side we need to work on fully automated deductive verification based on decidable logics with both temporal and probabilistic features, and on the other side we need to work on improving the model checking ability to handle large state spaces (e.g., symbolic model-checking [9], probabilistic model checking [2], etc.).

Important work that seems neglected by the scientific community is the stabilization science, which provides a common approach to studying system stability through stability analysis [1, 6, 20]. In this approach, a system is linearized around its operating point to determine a small-signal linearized model of that operating point. The stability of the system is then determined using linear system stability analysis methods such as Routh-Hurwitz, Root Locus, Bode Plot, and Nyquist Criterion.

Stability analysis is the theory of validating the existence of stable states presented through differential equations that govern the system dynamics. Although, theoretically, there is no guarantee for the existence of a solution to an arbitrary set of nonlinear differential equations [5], we may use stabilization science to build small-signal linearized models for the different system components, anticipating that the linearized models of system components will yield a relatively small state space, enabling for their efficient verification [10]. Then we may apply compositional verification techniques to produce an overall system-wide verification.

Other, not that well-developed verification techniques are those related to automatic test-case generation and simulation [16], which may reduce testing costs and improve the quality of testing. For example, test cases can be generated from a formal specification of a system built with a domain-specific formal language. If combined with code generation and analysis techniques for efficient test-case generation (e.g., change-impact analysis), automatic test-case generation might be used to efficiently test system behavior under simulated conditions [16].

Moreover, high-performance computing can be used for parallelizing simulations, which will allow multiple state space explorations to occur simultaneously.

9 Conclusion

Any AI system is a subject to uncertainty due to potential evolution in execution environment, in requirements, business conditions, available technology, and the like. Thus, it is important to capture and plan for uncertainty as part of the development process. Failure to do so may result in systems that are overly rigid for their purpose, an eventuality of particular concern for domains that typically use AI, such

as unmanned space flight. Contemporary formal verification techniques can be very helpful in verifying safety properties via the formalization of non-desirable system states along with the formalization of behavior that will never lead the system to these states. Although complete safety is obviously not possible, the use of both deductive and probabilistic guarantees may eventually help us cover a wide range of the uncertainty in the AI systems' behavior. The current verification toolset is not powerful enough to guarantee safety in the complex AI behavior. Further enhancement of that toolset can be achieved by developing better automated reasoning and model checking, along with development of new verification techniques based on stabilization science, test-case generation and simulation. High-performance computing can be used for parallelization of the verification process.

Acknowledgements This work was supported with the financial support of the Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Research Centre (www.lero.ie).

References

1. Arora, A.: Stabilization. In: Encyclopedia of Distributed Computing. Kluwer Academic Publishers (2000)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
3. Berger, J.O.: Statistical Decision Theory and Bayesian Analysis. Springer Series in Statistics (Second ed.), Springer-Verlag (1985)
4. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5), 1512–1542 (1994). DOI 10.1145/186025.186051. URL http://doi.acm.org/10.1145/186025.186051
5. Conley, M., Appleby, B., Dahleh, M., Feron, E.: Computational complexity of Lyapunov stability analysis problems for a class of non-linear systmes. Society of industrial and applied mathematics journal of controland optimization **36**(6), 2176–2193 (1998)
6. Emadi, A., Ehsani, M.: Aircraft power systems: Technology, state of the art, and future trends. Aerospace and Electronic Systems Magazine **15**(1), 28–32 (2000)
7. Keeney, T.: Autonomous vehicles will reduce the chances of dying in an auto accident by over 80% (2015). ARK Analyst, url:http://ark-invest.com/industrial-innovation/autonomous-vehicles-will-reduce-auto-accidents
8. M. Hazewinkel, e.: Logical calculus. In: Encyclopedia of Mathematics. Springer (2001)
9. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers Norwell, MA, USA (1993)
10. Pullum, L., Cui, X., Vassev, E., Hinchey, M., Rouff, C., Buskens, R.: Verification of adaptive systems. In: Proceedings of (Infotech@Aerospace) Conference 2012, Garden Grove, California, USA, pp. 2012–2478. AIAA (2012)
11. Sternberg, R.J., Sternberg, K., Mio, J.: Cognitive Psychology, 6th Edition. Wadsworth Publishing (2012)
12. Vassev, E., Hinchey, M.: Awareness in software-intensive systems. IEEE Computer **45**(12), 84–87 (2012)
13. Vassev, E., Hinchey, M.: Autonomy requirements engineering. IEEE Computer **46**(8), 82–84 (2013)
14. Vassev, E., Hinchey, M.: Autonomy Requirements Engineering for Space Missions. NASA Monographs in Systems and Software Engineering. Springer (2014). DOI 10.1007/978-3-319-09816-6. URL http://dx.doi.org/10.1007/978-3-319-09816-6

15. Vassev, E., Hinchey, M.: Knowledge representation for adaptive and self-aware systems. In: Software Engineering for Collective Autonomic Systems, Volume 8998 of LNCS, pp. 221–247. Springer (2015)
16. Vassev, E., Hinchey, M., Nixon, P.: Automated test case generation of self-managing policies for NASA prototype missions developed with ASSL. In: Proceedings of the 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE2010), pp. 3–8. IEEE Computer Society (2010)
17. Vassev, E., Sterritt, R., Rouff, C., Hinchey, M.: Swarm technology at NASA: Building resilient systems. IT Professional **14**(2), 36–42 (2012)
18. Vinge, V.: The coming technological singularity: How to survive in the post-human era (1993). Https://www-rohan.sdsu.edu/faculty/vinge/misc/singularity.html
19. Wirsing, M., Holzl, M., Koch, N., Mayer, P. (eds.): Software Engineering for Collective Autonomic Systems, Volume 8998 of LNCS. Springer (2015)
20. Yerramalla, S., Fuller, E., Mladenovski, M., Cukic, B.: Lyapunov analysis of neural network stability in an adaptive flight control system. Self-Stabilizing Systems pp. 77–91 (2003)