Verifying Autonomous Systems

Michael Fisher
Dept. Computer Science
University of Liverpool
Liverpool, U.K.
mfisher@liverpool.ac.uk

Louise Dennis
Dept. Computer Science
University of Liverpool
Liverpool, U.K.
l.a.dennis@liverpool.ac.uk

Matt Webster
Virtual Engineering Centre
Daresbury Laboratory
Warrington, U.K.
matt@liverpool.ac.uk

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model Checking*

General Terms

Formal verification, hybrid architectures, autonomous systems

Keywords

Verification, autonomy, reliability, agents

1. INTRODUCTION

In this article we consider the question: "how should autonomous systems be analyzed?". In particular, we describe how the confluence of developments in two areas, *autonomous systems architectures* and *formal verification for rational agents*, can provide the basis for the formal verification of autonomous systems behaviours.

We discuss an approach to this question that involves

1. modelling the behaviour and describing the interface (input/output) to an *agent* in charge of making decisions within the system;
2. model checking the agent within an unrestricted environment representing the real world and those parts of the systems external to the agent, in order to establish some property, φ;
3. utilizing theorems or analysis of the environment, in the form of logical statements (where necessary), to derive properties of the larger system; and
4. if the agent is refined, modify (1), but if environmental properties are clarified, modify (3).

We begin with an overview of autonomous systems, their core attributes and uses, and the autonomous software that (typically) controls them. Such systems are now being deployed in safety, mission, or business critical scenarios, which means a thorough analysis of the choices the core software might make becomes crucial. But, should the analysis and verification of autonomous software be treated any differently to 'traditional' software used in critical situations? Or is there something new going on here? Before we tackle all these questions, we will first recap *autonomous systems* (in Section 2) and *formal verification* (in Section 3).

Submission for Review Article
ACM Communications.

2. AUTONOMY

Autonomous systems are systems that *decide for themselves* what to do and when to do it. Such systems might seem futuristic, but they are closer than we might think. Modern household, business, and industrial systems increasingly incorporate autonomy. There are many examples, all varying in the degree of autonomy used, from almost pure human control to fully autonomous activities with minimal human interaction. Application areas are broad, ranging from health-care monitoring to autonomous vehicles.

But what are the reasons for this increase in autonomy? Typically, autonomy is used in systems that:

1. must be deployed in *remote* environments where direct human control is infeasible;
2. must be deployed in *hostile* environments where it is dangerous for humans to be nearby, and so difficult for humans to assess the possibilities;
3. involve activity that is too lengthy and/or repetitive to be conducted successfully by humans; or
4. need to react *much* more quickly than humans can.

However, it may actually be *cheaper* to use an autonomous system. After all, humans need training, monitoring, safe environments, medical support, legal oversight, etc.

2.1 Examples

There are many autonomous systems that have either been deployed or are in development. We clearly cannot survey them all so only provide a broad selection below.

Robotics and Robot Swarms. As we move from the restricted manufacturing robots seen in factories towards robots in the home and robot helpers for the elderly, so the level of autonomy required increases.

Human-Robot Teamwork. Once we move beyond just directing robots to undertake tasks, they become robotic companions. In the not too distant future, we can foresee teams of humans and robots working together but making their decisions individually and autonomously.

Pervasive Systems, Intelligent Monitoring, etc. As sensors and communications are deployed throughout our physical environment and in many buildings, so the opportunity to bring together a multiplicity of sensor inputs has led to autonomous decision-making components that can, for instance, raise alarms and even take decisive action.

Autonomous Road Vehicles. Also known as "driverless cars", autonomous road vehicles have progressed beyond initial technology assessments (e.g. DARPA Grand Challenges) to the first government-licensed autonomous cars [35].

As we can see from the above examples, autonomous systems are increasingly being used in safety/mission/business critical areas. Consequently, they need rigorous analysis. One traditional way to achieve this, at least in non-autonomous systems, is to use *formal verification.* While applying formal verification techniques to autonomous systems can be difficult, developments in autonomous system architectures are opening up new possibilities.

2.2 Autonomous Systems Architectures

Many autonomous systems, ranging over unmanned aircraft, robotics, satellites and even purely software applications, have a similar internal structure, namely layered architectures [23] as summarized in Figure 1. Although purely connectionist/sub-symbolic architectures remain prevalent in some areas, such as robotics [10], there is a broad realization that separating out the important/difficult choices into an identifiable entity can be very useful for development, debugging, and analysis. While such layered architectures have been investigated for many years [23, 3] they appear increasingly common in autonomous systems.

Notice how the system in Figure 1 is split into *real-world interactions*, *continuous control systems*, and *discontinuous control.* For example, a typical unmanned air system might incorporate an aircraft, a set of control systems encapsulated within an autopilot, and a high-level decision-maker that makes the key 'choices'. Once a destination has been decided, the continuous dynamic control, in the form of the autopilot, will be able to fly there. The 'intelligence' only becomes involved if either an alternative destination is chosen, or if some fault or unexpected situation occurs.

But what is this 'intelligent' decision-making component? In the past this has often been conflated with the dynamic control elements, the whole being described using a large, possibly hierarchical, control system, genetic algorithm, or neural network. However, architectures are increasingly being deployed in which the autonomous, 'intelligent' decision-making component is captured as an 'agent'.

2.3 Agents as Autonomous Decision Makers

The development and analysis of autonomous systems, particularly autonomous software, is different to 'traditional' software in one crucial aspect. In designing, analyzing, or monitoring "normal" software we typically care about

- *what* the software does, and
- *when* the software does it.

Since autonomous software has to *make its own decisions*, it is often vital to know not only what the software does and when it does it, but also

- *why* the software chooses to do it.

This requirement – describing why a system chooses one course of action over another – provides new entities to be analyzed. But how shall we describe these new entities? A very useful abstraction for capturing such autonomous behaviour within complex, dynamic systems turns out to be the concept of an *agent* [22]. Since the agent concept came into prominence in the 1980s, there has been vast development within both academia and industry [4, 14, 20, 34]. It has become clear that this agent metaphor is very useful for capturing many practical situations involving complex systems comprising flexible, autonomous, and distributed components. In essence, agents must fundamentally be capable of *flexible autonomous action* [38].

However, it turns out that the 'agent' concept on its own is still not enough! Systems controlled by neural networks, genetic algorithms, complex control systems, etc., can all act autonomously and thus be called agents, yet the reasons for their actions are often quite opaque. Because of this, such systems are very hard to develop, control and analyze.

So, the concept of a *rational* agent has become more popular. Again, there are many variations [9, 33, 39] but we consider this to be an agent which

> *has explicit* reasons *for making the choices it does, and should be able to explain these if necessary.*

Therefore, a rational agent can be examined to discover *why* it chose a certain course of action. Such agents are often programmed and analyzed by describing their *motivations* (for example, 'goals'), *information* (for example, 'knowledge'), and how these change over time; see Section 3.3. Rational agents can adapt their autonomous behaviour to cater for the dynamic aspects of their environment, their requirements and their knowledge. Typically, they can also modify their decision-making following interactions with their environment. The predominant form of rational agent architecture is that provided through the *Beliefs*, *Desires*, and *Intentions* (BDI) approach [33, 32]. Here, the *beliefs* represent the agent's (probably incomplete, possibly incorrect) information about itself, other agents, and its environment, *desires* represent the agent's long-term goals, and *intentions* represent the goals that the agent is actively pursuing.

Before we move on to consider how we might verify autonomous systems and, in particular the rational agent that makes the core decisions, we first recap formal verification. In particular we will motivate and outline the tools and techniques for the agent verification that we have developed.

3. FORMAL VERIFICATION

So, we are clear now that autonomous systems are important, that their key decision-making components can usefully be represented through the rational agent concept, and that their increasing use in critical areas means that a deep and comprehensive form of analysis will be desirable. These concerns have led us to use *formal logics* for describing the required properties of our rational agents and then *formal verification* techniques to analyze how well the actual agents match these requirements. Formal verification encompasses a range of techniques that use mathematical and logical methods to assess the behaviour of systems. The most common approach is to exhaustively assess all the behaviours of a system against a logical specification [13]. But, how do we logically specify what an agent should do? In particular, how do we specify what decisions an agent can make and what motivations it has for making those decisions?

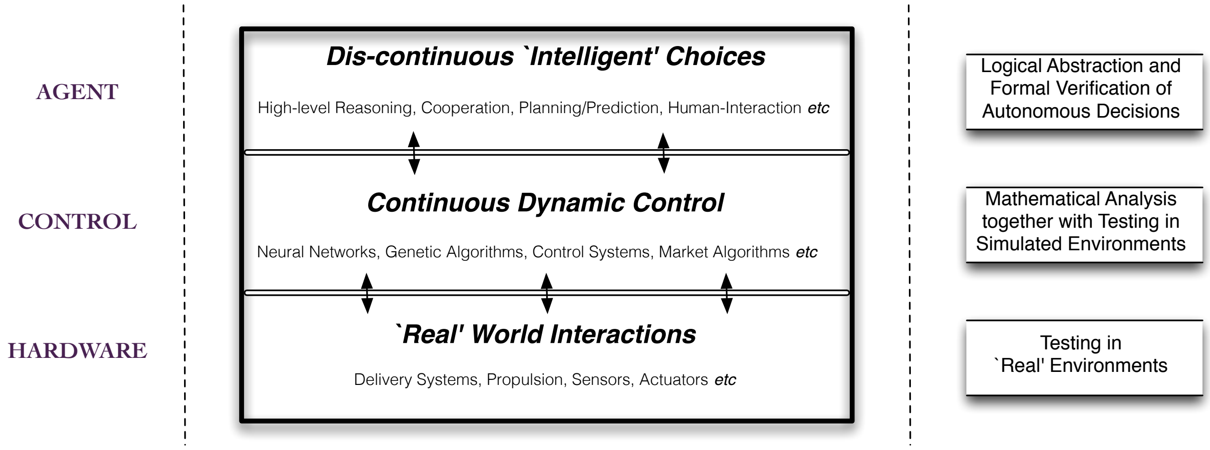

Figure 1: Typical Hybrid Autonomous System Architecture — With suitable analysis techniques noted.

3.1 Logical Agent Specification

Logics provide a well-understood and unambiguous formalism for describing the *behaviours*, *requirements*, and *properties* of systems. They have clear syntax and semantics, well-researched structural properties and comparative expressive power. Importantly, from our viewpoint, there are very *many* formal logics. This allows us to choose a logic appropriate to the types of properties and level of abstraction we require; for example:

- dynamic communicating systems $\longrightarrow$ *temporal logics*
- information $\longrightarrow$ *modal logics of knowledge*
- autonomous systems $\longrightarrow$ *modal logics of motivation*
- situated systems $\longrightarrow$ *modal logics of belief and context*
- timed systems $\longrightarrow$ *real-time temporal logics*
- uncertain systems $\longrightarrow$ *probabilistic logics*
- cooperative systems $\longrightarrow$ *cooperation/coalition logics*

So, we can usually choose logics that have the properties we require. Crucially, we can even construct new logics as the *combinations* of simpler logics. This turns out to be very useful for developing logical theories for rational agents as these typically consist of several dimensions:

Dynamism — temporal or dynamic logic;

Information — modal/probabilistic logics of belief or knowledge; and

Motivation — modal logics of goals, intentions, desires.

For example, the *BDI* approach combines [31]: a (branching) temporal/dynamic logic; a (KD45) modal logic of belief; a (KD) modal logic of desire; and a (KD) modal logic of intention. (For detail on different modal varieties, see [2].)

3.2 Formal Agent Verification

Once we have such a *logical requirement*, together with an *autonomous system* architecture wherein rational agent(s) encapsulate high-level decision-making, we have many options for carrying out formal verification, ranging across model-checking [13], runtime verification [24], and formal proof [21].

While there are also several approaches to agent verification [7, 29], the particular approach we adopt involves checking a BDI logical requirement against all practical executions of a program. This is termed the *model checking of programs* [36] and depends on being able to extract all these possible program executions, for example through *symbolic execution*. This contrasts to many model checking approaches in which an abstract model of the program must first be constructed before it can be checked against a property. In the case of `Java`, model-checking of programs is feasible as a modified *virtual machine* can be used to manipulate the program executions [26]. It is this last approach to agent verification we adopt. In order to do so, we must also give a very brief overview of agent programming languages.

3.3 Programming Rational Agents

We have seen how the rational agent approach provides the key model for describing autonomous decision-making. But, how can rational agents be programmed? Typically, programming languages for rational agents provide:

- a set of *beliefs*, representing information the agent has;
- a set of *goals*, representing motivations the agent has;
- a set of *rules/plans*, representing the agent's mechanisms for achieving goals;
- a set of *actions*, corresponding to the agent's external acts (delegated to other parts of the system); and
- deliberation mechanisms for deciding between alternative goals/plans/actions.

A typical agent rule/plan in such a language is:

```
Goal(eat) :   Belief(has_money), Belief(not has_food)
              <-   Goal(go_to_shop),
                   Action(buy_food),
                   Goal(go_home),
                   Action(eat),
                   +Belief(eaten).
```

The meaning of this rule is that, if the agent's goal is to 'eat' and if the agent believes that it has money but does not have food, then it will set up a new goal to go to the shop. Once that goal has been achieved, it will buy some food (delegated to a sub-system) and then set up a new goal to get home. Once at home it will eat and then update its beliefs to record that it believes it has eaten.

Such languages are essentially rule-based, goal-reduction languages, with the extra aspect that *deliberation*, the ability to change between goals and change between plan selection strategies at any time, is a core component. Almost all of these languages are implemented on top of `Java`, and the large number of agent platforms now available [5, 6] has meant that the *industrial* uptake of this technology is continually increasing. The key ancestor of most of today's agent programming languages is `AgentSpeak` [30], which introduced the programming of BDI agents using a modification of Logic Programming. Of the many descendants of `AgentSpeak`, we use GWENDOLEN [19], which is based upon *Jason* [8], for programming our rational agents. Consequently, it is such programs that we directly verify.

A full operational semantics for GWENDOLEN is presented in [15]. Key components of a GWENDOLEN agent are a set, Σ, of beliefs which are ground first order formulae and a set, I, of intentions that are stacks of *deeds* associated with some event. Deeds include (among other things) the addition or removal of beliefs, the establishment of new goals, and the execution of primitive actions. GWENDOLEN is event driven and events include the acquisition of new beliefs (typically via perception), messages and goals.

A programmer supplies plans that describe how to react to events by extending the deed stack of the intention associated with the event. The main task of a programmer working in GWENDOLEN is defining the system's initial beliefs and plans, these then describe the dynamic behaviour of the agent. A GWENDOLEN agent executes within a *reasoning cycle* which includes the addition of beliefs from perception, the processing of messages, the selection of intentions and plans, and the execution of deeds.

3.4 Model-Checking Agent Programs

We begin with *program model checking*, specifically the *Java PathFinder 2* system (JPF2), an open source explicit-state model checker for `Java` programs [36, 26]. Since the vast majority of agent languages are built on top of `Java`, we have extended JPF2 to the *Agent JPF* (AJPF) system [19] incorporating the checking of agent properties. However, in order to achieve this the semantics of the agent constructs used must be precisely defined. Such a semantics can be given using the *Agent Infrastructure Layer* (AIL) [16], a toolkit for providing formal semantics for agent languages (in particular BDI languages) built on `Java`. Thus, AJPF is essentially JPF2 with the theory of AIL *built in*; see [19].

The whole verification and programming system is called *MCAPL* and is freely available on Sourceforge[1]. As the model-checker is based on JPF2, the modified virtual machine is used to exhaustively explore all executions of the system. As each one is explored it is checked against the required property. If any violation is found, that execution is returned as a counterexample.

The GWENDOLEN language, mentioned above, is itself programmed using the AIL and so GWENDOLEN programs can be model checked directly via AJPF.

4. VERIFYING AUTONOMOUS SYSTEMS

Now we return to our original question: how do we go about *verifying* autonomous systems? Recall the architecture in Figure 1. For the 'traditional' parts there are well known, recognized, and trusted approaches, such as testing for real-world interaction and analytic techniques for continuous dynamics. But what about the agent that makes high-level, 'intelligent' choices about what to do? As we will explain next, it is our approach to use *formal verification* of the potential choices the agent can take. This is feasible since, while the space of possibilities covered by the continuous dynamics is huge (and potentially infinite), the high-level decision-making within the agent typically involves navigation within a discrete state space. The agent rarely, if ever, bases its choices directly on the *exact* values of sensors, etc. It might base its decision on values reaching a certain threshold, but relies on its continuous dynamics to alert it of this, and such alerts are typically binary valued (either the threshold has been reached or it has not). Thus, we propose the mixture of techniques in Figure 1 to provide the basis for formal verification of autonomous systems.

4.1 Verifying Autonomous Choices

How shall we verify autonomous decision-making? Our main proposal is to use program verification to demonstrate that the core rational agent always *endeavours* to act in line with our requirements and never *deliberately* chooses options that it believes will lead to bad situations (e.g. ones where the agent believes something is unsafe). Thus, we do not try to verify all the "real world" outcomes of the agent's choices, but instead verify the *choices* themselves. In particular, we verify that the agent always tries to achieve its goals/targets to the best of its knowledge/beliefs/ability. Thus, the agent achieves situations it *believes* to be good and avoids situations it *believes* to be bad. Consequently, any guarantees here are about the autonomous system's decisions, not about its overall effects.

This lets us distinguish between a rational agent *knowingly* choosing a dangerous/insecure option and a rational agent *unknowingly* doing so based on an imperfect representation of the actual environment. Indeed, we argue that the most crucial aspect of autonomous system verification, for example concerning safety, is to identify that the agent never deliberately makes a choice it *believes* to be unsafe. We wish to ensure that if an unsafe situation arises this is because of unforeseen consequences of an agent's actions (i.e. its model of the environment was too weak) not because the agent chose an option known to lead to a bad outcome.

Aside: Accidental or Deliberate?.

Are all dangerous situations equally bad? What if a robot *deliberately* took an action that it knew would cause danger? Is this more serious than a robot *accidentally* causing this danger? This distinction can be important, not least to the public, and if a robot is being 'vindictive', then few safeguards can protect us. Importantly, our approach allows us to distinguish between these cases. We can verify whether the agent beliefs were simply not accurate enough (in which case, the agent is 'innocent') or whether the agent knew about the danger and decided to proceed anyway.

One reason for our approach of verifying what the agent chooses, based on its beliefs, involves the purely practical issue of trying to model the "real world". We can *never* have a precise model of the "real world" and so can never

[1] `http://cgi.csc.liv.ac.uk/MCAPL`

say, for certain, what the effect of any action the system could choose might be. We might construct increasingly precise models approximating the "real world", but they can clearly never be perfect.

A second reason is to treat the agent, to some extent, as we might treat a human. In assessing human behaviour, we are happy if someone *is competent and tries their best* to achieve something. In particular, we consider someone as exhibiting 'safe' behaviour, if they have taken all the information they have access to into account and have competently made the safest decision they consider possible. Just as with humans, an agent's beliefs capture its partial knowledge about the "real world". The agent's beliefs might be wrong, or incorrect, **but** we only verify that the agent never *chooses* a course of action that it *believes* will lead to a bad situation. The agent's beliefs could be wrong and, of course, these beliefs might be refined/improved providing a better (more accurate) abstraction of the real situation.

We can contrast this with the traditional approach to formal (temporal) verification where we verify that bad things *never* happen and good things *eventually* happen. Instead, we only need verify that the agent *believes* these to be the case. This also has an impact upon the agent's selection of intentions/goals. As the agent is required to believe that no bad thing should occur, then it should never select an intention that it believes will lead to something bad, e.g.[2].

$$\mathbf{B}(\varphi \Rightarrow \Diamond \mathsf{bad}) \;\Rightarrow\; \Box\neg\mathbf{I}\varphi$$

So, if the agent believes that achieving 'φ' eventually leads to something bad, it will never intend to undertake φ.

In the context of the verifications discussed in this paper we use the property specification language that is provided with AJPF [19]. This language is propositional linear temporal logic (PLTL), extended with specific modalities for checking the contents of the agent's belief base (**B**), goal set (**G**), actions taken (**A**) and intentions — goals which are associated with a deed stack — (**I**).

This approach is clearly simpler as we can carry out verification without comprehensive modelling of the "real world". Thus, we verify the (finite) *choices* the agent has, rather than all the "real world" effects of those choices. Clearly, some parts of an agent's reasoning are still triggered by the arrival of information from the real world and we must deal with this appropriately. So, we first analyze the agent's program to assess what these incoming *perceptions* can be and then explore, via the AJPF model checker, all possible combinations of these. This allows us to be agnostic about how the real world might actually behave and simply verify how the agent behaves *no matter what* information it receives. Furthermore, this allows us to use hypotheses that explicitly describe how patterns of perception may occur in reality. Taking such an approach clearly gives rise to a large state space because we explore all possible combinations of inputs to a particular agent. However it also allows us to investigate a multi-agent system in a compositional way. Using standard *assume-guarantee* (or *rely-guarantee*) approaches [25, 28], we need only check the internal operation of a single agent at a time and can then combine the results from the model checking using deductive methods to prove theorems about the system as a whole.

[2]Here, '**B**' means "the agent believes", '$\Diamond$' means "at some future moment in time", '$\Box$' means "at all future moments in time", and '**I**' means "the agent intends".

5. EXAMPLE SCENARIOS

To exemplify this approach, we review several different scenarios that have been implemented using Gwendolen and verified formally using AJPF [37, 17]. In all these examples, the distinction in Figure 1 is central. The agent makes a decision, passes on to the continuous control to implement the fine detail, and then monitors the activity. The agent only becomes involved again if a new situation is reached, if a new decision is required, or if the agent notices some irregularity in the way the continuous control is working.

5.1 RoboCup Rescue Scenario

Imagine an "urban search and rescue" scenario, of the form proposed in the *RoboCup Rescue* challenge [27], where autonomous robots are searching for survivors after some natural disaster (e.g. an earthquake). A robot builds up beliefs about some area using sensor inputs. Based on these beliefs, the robot makes decisions about whether to search further. So, we might verify [17]:

$$\Box(\mathbf{B}\,\mathsf{can\_leave} \Rightarrow (\mathbf{B}\,\mathsf{found}\;\vee\;\mathbf{B}\,\mathsf{area\_empty}))$$

meaning *if the searching robot believes it can leave the area, then it either believes a human is found or it believes the area is empty.* We can verify this, but need to provide some abstraction of the sensor inputs. We model the environment by supplying, randomly, all relevant incoming perceptions to the robot. In this case it either detects a survivor or does not detect a survivor, at any location. It is important to note that the robot could be *wrong.* Its sensors might not detect a survivor (e.g. buried under rubble). However, this does not make the autonomous system *incorrect*; it has made the best decisions it can given the information it had.

When AJPF encounters a random choice in Java it treats it as a branch in the possible execution of the model and explores both branches – i.e. it checks the property holds both in the situation where the perception was received by the agent and the situation where the perception was not received. We can extend this to proving properties given simple assumptions about the behaviour of the real world. These assumptions might be verified using other forms of analysis. Given the verification above, we might assume that the robot's sensors accurately detect the human, and that its motor control operates correctly. This allows us to prove a stronger property that the agent will *either* find the human or the area is *actually* empty. These deductive aspects can be carried out by hand, or by using a suitable prover.

In more sophisticated scenarios we may want to check properties of groups of systems/agents working together. Imagine that we now have another robot, capable of 'lifting' rubble. The two robots work as a team: the 'searching' robot will find the human, then the 'lifting' robot will come and remove the rubble. We will refer to the beliefs of the lifting robot as $\mathbf{B}_l$. Ideally, if these two work together as expected then we would like to show that eventually the lifter believes the human is free: $\Diamond\mathbf{B}_l\mathsf{free}(\mathsf{human})$. However, this depends on several things, for example that any communication between the robots is reliable. We can check the behaviours of each agent separately, then combine these component properties with statements about communication, in order to verify whether the robots can cooperate.

We have been verifying the beliefs agents form about their environment *in lieu* of verifying actual *facts*. However, some choices we may legitimately wish to verify depend upon the outcomes of previous choices being as expected. Suppose that our lifting agent does *not* deduce that the human is free (because it has moved some rubble), but continues to lift rubble out of the way until its sensors tell it the area is clear. We cannot verify that the robot will eventually believe the human is free since we cannot be sure that it will ever believe that the human is clear of rubble. However, we can establish (and have verified) that *assuming that, whenever the lifter forms an intention to free the human it will eventually believe the rubble is clear, then receipt of a message from the searching robot that a trapped human is located will eventually result in the lifter believing the human is free.*

$$\Box(\mathbf{I}_l \mathsf{free}(\mathsf{human}) \Rightarrow \Diamond \mathbf{B}_l \mathsf{clear}) \quad \Rightarrow \\ (\mathbf{B}_l \mathsf{receive}(\mathsf{searcher}, \mathsf{found}) \Rightarrow \Diamond \mathbf{B}_l \mathsf{free}(\mathsf{human}))$$

While much simplification has occurred here, it should be clear how we can carry out compositional verification, mixing agent model checking and temporal/modal proof. The input from sensors can be modelled in various ways to provide increasingly refined abstractions of the "real world". Crucially we can assess the choices the agent makes based on its beliefs about its environment and not necessarily what *actually* happens in its environment [3].

5.2 Autonomous Satellite Scenario

Consider a satellite orbiting the Earth and attempting to keep on a particular path [18]. We want to establish $\mathbf{B}\,\Box\mathsf{on\_path}$, i.e. the satellite believes it is always on the path. Yet, we cannot establish this since the satellite's agent cannot be sure that it will never leave the path (since this would be an impossibly strong assumption about the environment). However, we *can* show that

1. if it does leave its path, then the satellite will eventually recognize this, and
2. once this situation is recognized, the satellite will have a goal (i.e. 'intends') to move back onto the path as soon as possible.

In other words, if anything goes wrong, the satellite will recognize this and will *try* to fix it. It might fail, but all we can show is that it always tries to succeed. Note that (1) above is a property that needs to be established concerning the satellite's sensors, but (2) is indeed something we can verify of the agent.

Engineers and mathematicians have developed strong techniques for analyzing control systems and scenarios and 'proving' that a certain property holds. For example, we might separately prove that a continuous path planning algorithm works and so capture that as a behaviour in a simplified model of the environment (here, '**A**' means "the agent executes the external action of"):

$$\mathbf{A}\, \mathsf{go\_to\_path} \Rightarrow \ \Diamond \mathsf{on\_path}$$

[3]Agent code written in GWENDOLEN for this scenario together with sample verified properties is available from the MCAPL repository on Sourceforge.

Thus, if the agent executes some action based on continuous path planning to reach some destination it will eventually reach that destination. Again, notice how the verification of this will be carried out using other methods; we will just use this assumption during verification of the agent choices. As examples, we can verify several different properties [17] [3]:

1. Using a simple model of the environment where the satellite simply receives information about its position, we can verify that *if, whenever an agent uses continuous planning to move to a path, it eventually believes it reaches the path and if, whenever it activates path maintenance procedures it always believes it remains on the path, then eventually the satellite always believes it is on the path*:

$$\Box(\mathbf{A}\, \mathsf{go\_to\_path} \Rightarrow \Diamond \mathbf{B}\, \mathsf{on\_path}) \wedge \\ \Box(\mathbf{A}\, \mathsf{maintain\_path} \Rightarrow (\mathbf{B} \mathsf{on\_path} \Rightarrow \Box \mathbf{B}\, \mathsf{on\_path})) \\ \Rightarrow \Diamond \Box \mathbf{B}\, \mathsf{on\_path}$$

2. It is possible for venting from a broken fuel line to knock a satellite off path. In this situation the satellite first needs to correct the problem with the thruster (e.g. by switching valves between fuel lines) and then calculate a new path to its destination. So we can verify *if the satellite notices it is no longer on the path then it will form an intention to return to the path*:

$$\Box(\mathbf{B}\, \neg\mathsf{on\_path} \Rightarrow \Diamond \mathbf{I}\, \mathsf{on\_path})$$

Note. If the satellite receives a message requesting it to move to a different position during this process, then subtle interactions between the agent's goals and plans can result in the the satellite attempting to move to two locations at once. Attempting (and failing) to verify that, under suitable conditions, the agent would always eventually get on to the path led to the detection of a number of bugs such as this.

3. If we relax our hypotheses, for instance to allow the possibility of unfixable errors in the thrusters, then we can still verify some properties. For instance *eventually either the agent always believes it is on the path or it has informed ground control of a problem.*

$$\Diamond(\Box \mathbf{B}\, \mathsf{on\_path} \vee \mathbf{B}\, \mathsf{informed}(ground, problem))$$

5.3 Autonomous Unmanned Aircraft Scenario

Unmanned aircraft are set to undertake a wide variety of roles within civil airspace. For safety, and to obtain regulatory approval, unmanned aircraft must be shown to be *equivalent* to manned aircraft and *transparent* to other airspace users [12]. In essence, any autonomous systems in control of an unmanned aircraft must be "human equivalent" or better. Human equivalence is, clearly, difficult to specify. But perhaps a good place to start extracting desirable human behaviours are the statutory and regulatory documents designed to specify and exemplify ideal human behaviours, e.g. the "Rules of the Air" [11]. In order to begin to verify the human equivalence of unmanned aircraft autonomy, we identified a very small (but salient) subset of the Rules of the Air [37], including the following.

1. Detect and Avoid: "*. . . when two aircraft are approaching head-on .... and there is danger of collision, each shall alter its course to the right.*" (Section 2.4.10)

2. Navigation in Aerodrome Airspace: "*[An aircraft in the vicinity of an aerodrome must] make all turns to the left unless [told otherwise].*" (Section 2.4.12(1)(b))

3. Air Traffic Control Taxi Clearance: "*An aircraft shall not taxi on the apron or the manoeuvring area of an aerodrome without [permission].*" (Section 2.7.40)

A decision-making agent for an unmanned aircraft was written. A simulated environment was also developed using GWENDOLEN, consisting of: a sensor unit to generate alerts related to intruder aircraft and other air traffic; a navigation manager to generate alerts about the current flight path; and an aerodrome air traffic controller unit to simulate aerodrome air traffic control. In order to formally verify that the agent controlling the unmanned aircraft will follow the three rules above, they were translated into the logical formulae and verified using the AJPF model checker [37]:

1. *"It is always the case that if the agent believes that an object is approaching head-on, then the agent believes that the direction of the aircraft is to the right."*

$$\Box(\mathbf{B}\,\mathsf{objectIsApproaching} \Rightarrow \mathbf{B}\,\mathsf{direction}(\mathit{right}))$$

2. *"It is always the case that if the agent believes that it is changing heading (i.e. turning as part of navigation) and it believes it is near an aerodrome and it believes it has not been told to do otherwise, then the agent will not believe that its direction is to the right."*

$$\Box\left(\begin{bmatrix}\mathbf{B}\,\mathsf{changeHeading}\wedge\\ \mathbf{B}\,\mathsf{nearAerodrome}\wedge\\ \neg\mathbf{B}\,\mathsf{toldOtherwise}\end{bmatrix} \Rightarrow \neg\mathbf{B}\,\mathsf{direction}(\mathit{right})\right)$$

3. *"It always the case that if the agent believes it is taxiing, then it believes that taxi clearance has been given."*

$$\Box(\mathbf{B}\,\mathsf{taxiing} \Rightarrow \mathbf{B}\,\mathsf{taxiClearanceGiven})$$

Verifying such requirements not only shows that the autonomous system makes choices consistent with these "Rules of the Air", but can also highlight inconsistencies within the rules themselves [37].

6. SUMMARY

Once autonomous systems have a distinguished decision making 'agent', then we can *formally verify* this agent's behaviour. In particular, we have developed *model-checking* techniques for rational agents, allowing us to explore *all* possible choices the agent might make. Notably, the architecture and the logical framework together allow us to verify not only what the agent chooses but *why* it chooses it.

A central theme of our analysis of autonomous systems, and of the agents that control them, is to verify what the agent *tries* to do. Without a complete model of the real environment, then we cannot say that the system will *always* achieve something, but we can say that it will always *try* (to the best of its knowledge/ability) to achieve it. This is not only as much as we can reasonably say, but is entirely justifiable as we wish to distinguish accidental and deliberate danger. So, when considering safety, we cannot guarantee that our system will *never* reach an 'unsafe' situation, but we can guarantee the agent will never "knowingly" choose to move towards such a situation. Thus, all the choices of the agent/system are verified to ensure that it *never* chooses goals/actions which *it believes* will lead to 'bad' situations. Crucially, this analysis concerns just the agent's internal decisions and so verification can be carried out without having to examine details of the "real world". Thus, we verify the (finite) *choices* the agent has, rather than the (continuous/uncertain) real world effect of those choices.

Overall, we can see this as a shift from considering whether a system is "correct" to considering two aspects of systems:

1. analysis of *whether the (autonomous) system makes only "correct" choices, given what it believes about its environment*, together with

2. analysis of *how accurate and reliable the system's beliefs are about its environment.*

We have considered (1) in this article. However, (2) may be discrete, if abstractions are used, or continuous and uncertain, requiring more complex analytical techniques.

6.1 Future Work

This work is only just at the beginning, and the theme of verifying what autonomous systems *try* to do, rather than the effects they have, has much potential. However there are many avenues of future work, the foremost currently being incorporation of uncertainty and probability. So, rather than verifying *the agent never chooses a course of action that it believes will lead to a 'bad' situation*, we would like to verify that the agent

> *never chooses a course of action that it believes is more* likely *to reach a 'bad' situation than its other options.*

In addition, there are clearly various different forms of 'bad' situation, with different probabilities and measures concerning their seriousness. Again, these measures and probabilities should be incorporated into the properties verified.

Similarly there are important aspects of truly autonomous behaviour, such as the ability to plan and learn that we have not considered in any detail. We are interested in exploring how an agent might reason about new plans, for instance, to ensure that their execution did not violate any important properties and so provide guarantees about the agents overall behaviour even in the face of changing internal processes.

It is also important to assess if, and how, other approaches to the formalization of autonomous behaviours, e.g. [1], can be involved in our verification.

6.2 Towards Certification

Certification can be seen as the process of negotiating with a certain legal authority in order to convince them that relevant safety requirements have been explored and mitigated in an appropriate way. As part of this process, various items of evidence are provided to advance the applicant's safety argument. This approach is widely used for the certification of real systems, from aircraft to safety critical software.

Clearly, we are mainly concerned with the certification of *autonomous* systems. As above, systems might generally be analyzed with respect to the question, "Is it safe?" If there is a human involved at some point, e.g. a pilot or controller, then some view must be taken on whether the human acts to preserve safety or not. For example, within aircraft certification arguments, it is usually assumed that a pilot, given appropriate information and capabilities, will act to preserve the aircraft's safety. Yet in a safety analysis, we rarely go any further. Essentially, the human is assumed to be benevolent.

Our approach provides a mechanism for analyzing the agent choices in the case of autonomous systems. Thus, while a standard safety argument might skip over human choices, assuming the pilot/driver/operator will endeavour to remain safe, we can formally verify that the agent indeed tries its best to remain safe. In this way, our approach allows wider analysis — while the intentions and choices of a pilot/driver/operator must be assumed to be good, we can actually examine the intentions and choices of an autonomous system in detail.

Acknowledgments. This work was partially supported by EPSRC, while the Virtual Engineering Centre is a University of Liverpool project partially supported by both NWDA and ERDF. We are grateful to our many collaborators, but particularly Rafael Bordini, Neil Cameron, Mike Jump, Alexei Lisitsa, Nick Lincoln, Bertie Müller, and Sandor Veres.

7. REFERENCES

[1] R. Arkin. Governing Lethal Behavior: Embedding Ethics in a Hybrid Deliberative/Reactive Robot Architecture. Technical Report GIT-GVU-07-11, Georgia Tech, 2007.

[2] P. Blackburn, J. van Benthem, and F. Wolter, eds. *Handbook of Modal Logic*. Elsevier, 2006.

[3] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an Architecture for Intelligent, Reactive Agents. *J. Exp. Theor. Artif. Intell.*, 9(2–3):237–256, 1997.

[4] A. Bond and L. Gasser, eds. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, 1988.

[5] R. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, eds. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005.

[6] R. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, eds. *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009.

[7] R. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying Multi-Agent Programs by Model Checking. *J. Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[8] R. Bordini, J. Hübner, and M. Wooldridge. *Programming Multi-agent Systems in AgentSpeak using Jason*. Wiley, 2007.

[9] M. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.

[10] R. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE J. Robotics and Automation*, 2(10), 1986.

[11] Civil Aviation Authority. CAP 393 Air Navigation: The Order and the Regulations. `http://www.caa.co.uk/docs/33/CAP393.pdf`, April 2010.

[12] Civil Aviation Authority. CAP 722 Unmanned Aircraft System Operations in UK Airspace — Guidance. `http://www.caa.co.uk/docs/33/CAP722.pdf`, April 2010.

[13] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[14] P. Cohen and H. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42:213–261, 1990.

[15] L. Dennis and B. Farwer. Gwendolen: A BDI Language for Verifiable Agents. In *Workshop on Logic and the Simulation of Interaction and Reasoning*. AISB, 2008.

[16] L. Dennis, B. Farwer, R. Bordini, M. Fisher, and M. Wooldridge. A Common Semantic Basis for BDI Languages. In *Proc. 7th Int. Workshop on Programming Multiagent Systems*, *LNAI 4908*, pages 124–139. Springer, 2008.

[17] L. Dennis, M. Fisher, N. Lincoln, A. Lisitsa, and S. Veres. Verifying Practical Autonomous Systems. (Under review.)

[18] L. Dennis, M. Fisher, A. Lisitsa, N. Lincoln, and S. Veres. Satellite Control Using Rational Agent Programming. *IEEE Intelligent Systems*, 25(3):92–97, May/June 2010.

[19] L. Dennis, M. Fisher, M. Webster, and R. Bordini. Model Checking Agent Programming Languages. *Automated Software Engineering*, 19(1):5–63, 2012.

[20] E. Durfee, V. Lesser, and D. Corkill. Trends in Cooperative Distributed Problem Solving. *IEEE Trans. Knowledge and Data Engineering*, 1(1), 1989.

[21] M. Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic*. Wiley, 2011.

[22] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III*, *LNCS 1193*, pages 21–35. 1996.

[23] E. Gat, R. Bonnasso, R. Murphy, and A. Press. On Three-Layer Architectures. In *Artificial Intelligence and Mobile Robots*, pages 195–210. AAAI Press, 1997.

[24] K. Havelund and G. Rosu. Monitoring Programs Using Rewriting. In *Proc. 16th IEEE Int. Conf. Automated Software Engineering (ASE)*, pages 135–143. IEEE Computer Society Press, 2001.

[25] C. Jones. *Systematic Software Development Using VDM*. Prentice Hall International, 1986.

[26] Java PathFinder. `javapathfinder.sourceforge.net`.

[27] H. Kitano and S. Tadokoro. RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine*, 22(1):39–52, 2001.

[28] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[29] F. Raimondi and A. Lomuscio. Automatic Verification of Multi-agent Systems by Model Checking via Ordered Binary Decision Diagrams. *Journal of Applied Logic*, 5(2):235–251, 2007.

[30] A. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, *LNCS 1038*, pages 42–55. Springer, 1996.

[31] A. Rao. Decision Procedures for Propositional Linear-Time Belief-Desire-Intention Logics. *Journal of Logic and Computation*, 8(3):293–342, 1998.

[32] A. S. Rao and M. Georgeff. BDI Agents: From Theory to Practice. In *Proc. 1st Int. Conf. Multi-Agent Systems*, pages 312–319, San Francisco, USA, 1995.

[33] A. S. Rao and M. P. Georgeff. An Abstract Architecture for Rational Agents. In *Proc. 1st Int. Conf. Knowledge Representation and Reasoning*, pages 439–449, 1992.

[34] Y. Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[35] United States of America State of Nevada Legislature. Nevada Revised Statutes Chapter 482A — Autonomous Vehicles, March 2012.

[36] W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.

[37] M. Webster, M. Fisher, N. Cameron, and M. Jump. Formal Methods and the Certification of Autonomous Unmanned Aircraft Systems. In *Proc. 30th International Conference on Computer Safety, Reliability and Security*, *LNCS 6894*, pages 228–242. Springer, 2011.

[38] M. Wooldridge. *An Introduction to Multiagent Systems*. Wiley, 2002.

[39] M. Wooldridge and A. Rao, eds. *Foundations of Rational Agency*. Kluwer Academic Publishers, 1999.