Impossibility Results for the Online Verification of Ethical and Legal Behaviour of Robots

Jan van Leeuwen

Jiří Wiedermann

Technical Report UU-PCS-2021-02
April 2021

Center for Philosophy of Computer Science
Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

Series: UU-PCS

Department of Information and Computing Sciences
Utrecht University
Princetonplein 5
3584 CC Utrecht
The Netherlands

Impossibility Results for the Online Verification of Ethical and Legal Behaviour of Robots*

Jan van Leeuwen[1] Jiří Wiedermann[2]

[1] Dept. of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, the Netherlands
J.vanLeeuwen1@uu.nl

[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

"In the near term, one natural way to think about minimizing risk of harm from robots is to program them to obey our laws or follow a code of ethics."
Lin, Abney, and Bekey [24], 2011, p. 946

Abstract. We consider scenarios in which autonomous robots are expected to interact in accordance with certain rules of law or ethics, or any other set of formally expressed constraints. Can an 'observer' actually tell from inspecting a robot's program and interactions whether the robot will always follow the rules it is said to obey in its interactions with other robots? We argue that, under reasonable assumptions about robot programming and about an observer's capabilities, no (deterministic) module, algorithmic or otherwise, will enable an observer to do so for all robots in any feasible way. It means that, in general, guarantees about the legal or ethical behaviour of autonomous robots are not verifiable at runtime by any means, given the assumptions we made. The result holds for all 'non-trivial' robot interaction properties.

Keywords: autonomous robots, computability theory, cyber-physical systems, impossibility, machine ethics, observer dependence, robot law, verification.

1 Introduction

Autonomous robots are programmed to deal with complex situations with minimal or no human intervention. The robots are increasingly applied in advanced products like self-driving cars and in smart manufacturing. It is commonly envisioned that, in the not-too-distant future, autonomous robots will be interacting with other robots and with humans on a large scale, aiming to achieve any desirable effect in the environment in which they operate.

The promises of autonomous robot technology are daunting. The robots are often viewed as intelligent machines that can be programmed to perform, or learn, any task that is normally carried out by humans. The complexity of the tasks and situations they are able to handle on their own is steadily increasing. However, so are the potential safety risks when deploying them [16, 30].

* Version dated: April 30, 2021. This work was partially supported by ICS CAS fund RVO 67985807, CAS Programme Strategy 21, and the Karel Čapek Center for Values in Science and Technology (Prague, Czech Republic).

In order to prevent autonomous robots from causing any harm, to humans and to themselves, the governing software of these robots should guarantee that they adhere to the desired norms and moral standards, keeping them from taking any actions that violate these standards 'by design' [12]. In particular, the robots should be programmed such that they obey accepted rules of law and ethics during their operation [4, 9, 23, 24, 36] at any time. The all-important question that arises is: *can robot software of this quality be constructed?*

It is generally difficult to determine whether any software system adheres to its specification in all respects. Focusing on robotic systems now, can one be certain of any presumed moral behaviour of these systems if this is specified in their design? Note that the 'moral layer' of any autonomous robot system is likely to involve an implementation of the known and specified moral rules that apply and a reasoning system for them, and possibly a learning mechanism to expand the rule base during the operation of the system as well [6, 28, 41]. When a robot is said to obey the rules of law or ethics, what guarantee do we actually have that it does so at any time. *Can the robot be trusted?*

Verification Legal and ethical requirements clearly extend, and complicate, the correctness issue for robot programs. Assuming that the requirements are 'formally specified' for the purpose of software engineering, it has to be verified that they are implemented *correctly* and that they are *complete*, i.e. applicable to all circumstances that a robot can encounter [10, 26]. Will a robot always act in a legally or ethically responsible manner as expected, even when restricted to some well-delineated domain?

To investigate this question, we assume a simple model of *robot verification*. The model consists of an *observer* which can inspect a robot program and monitor the robot's behaviour under all circumstances in practice. When the observer disagrees with the observed behaviour of the robot, we assume that a 'problem' has been detected and that the robot program can not be certified as being correct. It leads to the central question of this report:

> *Can an observer always tell from inspecting and monitoring a robot's program whether the robot will always obey the given rules of law or ethics, or any other set of formally expressed constraints, in any interaction with other robots (or humans)?*

The answer will prove to be *no*, in general (or at least, not within feasible resource bounds). More precisely, under reasonable assumptions about robot programs and an observer's capabilities, no realistic (deterministic) *module*, algorithmic or otherwise, will enable an observer to tell whether a robot always acts correctly or not. It means that in general, under suitable assumptions, legal or ethical guarantees about the interactive behaviour of autonomous robots are *not* verifiable by any kind of classical algorithm. The result holds for *all* robot properties that are *non-trivial*, in a sense that will be defined later.

Reflections From a computability-theoretic viewpoint, the result does not come as a surprise. By *Rice's theorem* [33], any non-trivial semantic property of computer programs is undecidable. Thus, if Rice's theorem could be applied to robot programs, compliance to a given set of rules of law or ethics would be

'undecidable' for them, just like any other non-trivial semantic property. However, *robot programs are not classical programs as formalized in computability theory*. For example, unlike classical ones, robot programs operate interactively and always produce an output action in finite time, for any situational input they keep on receiving. Let's consider some further aspects of the results we intend to prove.

a) Robots Interestingly, Kant believed that ethics is 'decidable', in the sense that every ethical/moral problem would have only one correct resolution (with various criteria such as 'universalizability') [32]. (This is not the same notion of decidability as we know it in computability theory.) However, what if we consider the claim for the behaviour of *robots*, e.g. operating in an *ensemble* with other autonomous robots? Can it always be decided, based on a robot's program, whether a robot will act ethically or not, in all situations that it can encounter in the ensemble? Can an observer always tell, also when it comes to determining how robots act on *ethical dilemmas* (in the observer's view)?

b) Impossibility In order to derive a generic result similar to Rice's theorem in computability theory, we will define a general framework for robot programming that allows us to understand how complex robot programs can be, and thus, how complex a task it can be for an observer to verify them. We will then show that, if an observer had a (deterministic) *module* to determine whether a robot always complies with a given set of rules of law or ethics (or not, according to the observer) during its interactions, then a robot can be designed for which realistic interactive situations exist in which the module must fail. This contradicts the assumption that the module could always determine, when called, whether the robot would comply or not comply.

Strictly speaking, our result is an *impossibility result* rather than an 'undecidability' result: it shows that, in the broad framework that will be defined and given any non-trivial set of legal or ethical constraints, no deterministic module can correctly verify the legal or ethical behaviour of a robot in all situations, based on the robot's program alone. The result does not even assume that the module is necessarily algorithmic.

c) Machine ethics The question we consider is relevant for the formal verification of (programmed) moral behaviours of autonomous systems in all of AI [7–9], especially in *machine ethics* [1, 4, 41]. The latter field concerns the ethical principles that are implemented in (the programs of) artificial agents and applied in their interactions with other agents, notably other autonomous robots and humans. It is non-trivial to argue that the term 'ethics' is indeed justified in the context of artificial agents [19, 25, 29]. A well-known example of a set of principles is given by Asimov's *Three Laws of Robotics* [2, 22, 39], but many other examples can be imagined, depending on the specific context in which a robot operates and the regulations that apply [21, 31].

d) Complexity Complete verification of program specifications is computationally hard in general, and this appears to be no different when restricted to the legal or ethical specifications of robot programs [6, 10, 23, 26, 27, 35]. Charisi *et al.* [8] note that *'full formal verification is likely to be unrealistic [...] both because of non-symbolic components and because of practical complexity'*. Yet

there seem to be only a few theoretical results concerning the verification of legal or ethical constraints for robot programs. For example, Englert *et al.* [14] construct a somewhat special problem in which the ethical 'correctness' of an autonomous robot's decision depends on its ability to resolve the *Halting Problem* of another agent's program. This will not be algorithmically possible in general, as the halting problem for computer programs is undecidable [33]. (The argument has been discussed from a practical point of view in [18].)

We prove a general result that, given any non-trivial set of constraints to be monitored, purely addresses the possibility or impossibility of a module (e.g. an algorithm) for the observer that will enable the observer to determine whether (the program of) an autonomous robot always decides correctly in any situation that the robot may encounter, based solely on its monitoring of the program of the observed robot. The result we prove includes both the practical and theoretical impossibility.

Outline The remainder of this report is organized as follows. In Section 2 we describe our general assumptions about autonomous robots and their interaction, and about the kind of programs that control them. In Section 3 we introduce a plausible scenario for how an observer might 'check' the behaviour of an autonomous robot, and define what it means for a *robot property*, like that of producing only legal or ethical actions, to be *non-trivial*. As a first result we show that, when robots can resort to *unbounded* memory, no algorithmic procedure can exist that will enable an observer to tell whether the actions of a robot always satisfy P, where P is any non-trivial robot property.

In Section 4 we refine the scenario, include the possibility that robots only have *bounded* memory (in terms of the size of the observed programs), and define what it means for an *interaction property* to be non-trivial. In Section 5 we then prove the main result, showing that, for *any* non-trivial property P, no deterministic *module* of any kind can exist that will enable the observer to determine whether a given robot (program) always acts 'correctly' during its interactions with another robot, i.e. while satisfying property P, under all circumstances. In Section 6 we discuss the result and the various assumptions we made to prove it. Finally, in Section 7 we offer some conclusions.

2 Programming model

In order to claim that certain robot properties are impossible to verify, we need to be precise about the programs and the scenarios we allow. In this section we summarize the basic assumptions about robots and robot programming. We sketch a general model of groups of interacting autonomous robots, and describe how the behaviour of the robots in this setting is supported by suitable constructs in their program.

2.1 On robots and their programs

Autonomous robots will be viewed as *cyber-physical systems*. Following [37], we assume that the robots have a finite number of processors, sensors, actuators, and effectors that enable them to monitor themselves and their surroundings,

and to operate in their *environment*. The actions of a robot at any moment are assumed to be determined by its program and its so-called *cognitive state*, which consists of all important variable data of the system such as the state of its processors and the signals and controls received from and sent to its sensory and motor units at that moment, respectively. All system data is assumed to be *finitary*. Given a robot A, we distinguish between the'internal' state information (q_A) of the robot, and the 'external' information (S_A) that it senses about the *situation* in its environment (which may depend on the way the robot senses, but not on its internal state).

Programs Robot programs use the input obtained from a robot's sensors and effectors (signals, sensory information, and feedback reports) to update the cognitive state of the robot and to compute and direct its subsequent actions, i.e. motions and interactions, towards achieving a certain task. We assume that every robot only interacts with a bounded number of other robots at a time, that it is distinguished by a unique identity (used for this purpose only), and that at every moment it can 'read' some, or all of the state information of every robot it is interacting with.

We assume that autonomous robots can move around, interact, face 'situations', and act in their own way in the environment in which they operate. We are especially interested in notable properties of their interactions, e.g. when one robot influences another robot in a particular way (in a certain situation). We say that robot A *interacts with* robot B if A and B know of each other's presence and A can send messages to B (and usually vice versa). We will say that A *targets* B if A and B know of each other's presence and A specifically 'intends' to send some kind of 'disrupting message' to B, if and when its program tells it to. Targeting is an important special case of interaction, and we assume that it is recognizable when a robot targets another one.

Robot software consists of many cooperating processes, but we will focus on the chief control program that controls and directs the actions of the robot (agent) as an entity. Robots with the same control programs are said to be of the same *sort* and we use this term implicitly to refer to their program as well. We make the following concrete assumptions about robot programs, their control structures and their operation.

Situations We assume that robots maintain some kind of representation of their (physical) environment, which enables them to keep track of all relevant objects in it (including other robots). Given a robot A, the information about the current (external) situation as it perceives it, is represented in its cognitive state by S_A, a structured constant or variable that captures the, *finitely many*, determining aspects of its surroundings that are relevant for its effective handling. Processing (the values in) S_A at a certain moment may involve all kind of modellings and subroutines that use the sensed data. We assume that, once it is set, S_A remains valid long enough for A, and for any robot C that is interacting with it, to take appropriate actions based on it (if any). In the sequel we will identify 'S_A' and the situation it represents.

A robot may run into situations of many different kinds, and all may lead to their own specific response. It may well happen that a robot must make a choice

between several different options (actions), with each choice having potentially different effects. The actual choice will be computed according to the robot's program when a concrete response must be determined, depending on the other inputs at the time and any further criteria.

Example 1. In many situations, a robot will have the option to move forward, to pass left or right, to turn a corner, or to stop. Each option may come with more or less severe consequences if chosen, depending on the obstacles in the vicinity. In the well-known Trolley problem [40], the option to turn a corner or to stop does not always exist, making it a philosophical challenge to program a robot such that it always decides for the 'right' action, in every situation.

Basic data and variables Every robot A of a given sort is assumed to maintain variables for the following information: its current internal state q_A, the sort and current internal state of every robot B that it can currently interact with and/or target, the sort of every robot C it can currently be targeted by, the data that characterizes its situation S_A, and a *copy* of the data that characterizes the situation S_B of every robot B that it can currently target. (One may well assume that A has access to this information for *all* robots it can interact with, but we won't need or use this in our model.) In addition, robot A has variables to hold the information it receives from, or will send to, the respective robots and to any candidates for future interactions. We assume that by knowing all this information, A can update its cognitive state and, in particular, infer which robots B it should 'interact with' or 'target', and how. We may assume w.l.o.g that every robot keeps a record of the robots it can interact with, target or could be targeted by at any time

H-list We assume that, in each application of our framework, there is a given *recursive* set H of *exceptional situations*, represented in the format utilized by our robots, that induce the following behaviour. Suppose that some robot B, in internal state q_B, is facing a situation $Z = S_B$ which is recognized as belonging to H. Then any robot A that can target B in this case will, if and when it actually targets B, target it with a message that *only* depends on B, q_B, and Z. Thus, the action of A that is triggered towards B in this case, is fully determined by the information in S_A that pertains to the 'exceptional circumstance' that it sees, without resort to any other variable data kept by A. (In contrast, when $Z \notin H$, A's action may well depend on *all* information in A's cognitive state.)

It will depend on the application which (classes of) situations S with the stated property are actually declared to be 'exceptional'. However, as a common feature we will require that, whenever a robot B is facing one of these situations, it potentially has a choice between actions that are and actions that are *not* preferred, in the view of the observer, with the actual choice depending on B's cognitive state at the time. Thus, exceptional situations always require robots to make *dichotomous decisions*.

As H is assumed to be recursive, the elements of H as well as those of its complement can be effectively enumerated, even without repetitions [33]. However, this will usually not be a very efficient way of deciding whether a given situation Z belongs to H or not. More typically, H will be given by means of a effective *routine* that tries to match a situation $Z = S_B$ that is sensed by a robot

B to one that fits (the description of) one of finitely many *exemplary situations* which are considered to be 'representative' for the exceptional situations in H. Note that situations are assumed to be finitary, and we may thus assume that they can be subjected to algorithmic analysis and classification.

Note that, when A targets robot B with $Z = S_B \in H$ at a certain moment, then A knows the data needed to determine its targeting message, namely B's sort (i.e. program), its current q_B, and $Z = S_B$. Notice that different robots A may well send different messages to B in this scenario.

Example 2. An H-list could consist e.g. of the situations that can occur in a medical operating room, in the cabin of a spacecraft in orbit, when approaching a traffic light at an intersection, in instances of the *Trolley problem* [40], or in a combination of any of these. It is easily seen that in all these cases, any robot B that is facing any of these situations can choose between good and bad actions, and that any robot A that targets such a robot B can potentially influence it to take one action or the other in a way that only depends on its knowledge of B, q_B and $Z = S_B$.

Operational cycle - general The program of every autonomous robot is assumed to consist of a basic *Sense-Plan-Act* cycle, following one of the standard robotic paradigms (cf. [37]). Concretely, the program of any robot A is assumed to consist of an initialization part $init_A$, followed by a cycle of three consecutive parts: $sense_A$, $plan_A$, and act_A. The cycle is repeated over and over, as long as the robot is 'on'. Before specifying this further, we make some general remarks about its programmatic realization.

Clearly, as a robot will be scanning and analysing various other robots and their situations simultaneously, its program may well have to consist of several different commands in the $plan_A$-part that may be 'triggered' simultaneously, with further criteria to choose between them or sequence them at runtime. Thus, we assume that the (entire) $plan_A$-part is structured as, what we call, a *hyper-command*.

Definition 1. *A hyper-command is a program construct consisting of finitely many concurrent* command blocks *and a* supervisory control *that determines (chooses) which command block(s) should be executed in a given iteration, i.e. at runtime.*

The command blocks correspond to the different lines of action which a robot must keep under consideration simultaneously.

For our arguments we do not need to know how, and by what criteria, a robot (program) actually makes its choices in the $plan_A$-part of the program, as long as each command block that *could* be triggered has a non-zero probability of being chosen (or, included). For example, when a certain robot B is in reach, and robot A can target it according to its hyper-command, then it must be possible for A to target B any time this happens and thus to execute the command(s) it has for it. Hyper-commands generalize so-called *guarded commands* [13], as implemented in various robot programming languages.

Operational cycle - parts During every iteration of its *Sense-Plan-Act* cycle, a robot has to process ('digest') the newly sensed information (in $sense_A$), 'compute' the actions that A could take and choose between them (in $plan_A$),

and 'execute' the selected action or sequence of actions (in act_A). The three parts of the operational cycle are dedicated as follows:

- $sense_A$: in each iteration of $sense_A$, robot A determines and/or updates the values of all its variables for which it has new data. In particular, it updates its situation data and the list of robots it can interact with, target or be targeted by (and their current state and situation data).
- $plan_A$: in this part, robot A matches the information it got with any backed-up or stored data from previous situations and interactions, and determines (computes) the action(s) it intends to take or continue. Based on whatever considerations, it leads to a choice of the command block in A's program that triggers the appropriate action. More concretely, the $plan_A$-part is a hyper-command composed of the following types of command blocks:
  - *type I* ('targeting commands'): any command block that may be executed when A *targets* a particular robot B in its vicinity. The action it computes consists of a well-determined 'special instruction' that A wants to send to B. We assume that the command block can only be of one of the following two *sub-types*, with $Z = S_B$ being the situation B is facing (and known to A by assumption):
    - *type I-a*: $Z \in H$ and, corresponding to the special ('exceptional') status of situations like Z, the action (instruction) that is computed *only* depends on B, q_B, and $Z = S_B$. In particular, it does not depend on any other variable data that A maintains.
    - *type I-b*: $Z \notin H$, and the action (instruction) that is computed may depend on all variable data that A maintains.

    Note: We assume that every robot program contains *at most one command block of type I-a* and *one of type I-b*. The command types will be discussed in more detail below. Recall that H is the *H-list*, assumed to be uniform for all programs in an application.
  - *type II* ('general computation and interaction commands'): any other command block in which A aims at acting on its own and possibly interacting with its environment in general, i.e. by computing an action to be taken, based on the current internal state q_A and current situation S_A (and any data A has available on other robots).
- act_A: finally, during the subsequent iteration of act_A, robot A carries out the concrete instructions for the action(s) it has computed, ranging from internal actions to be taken (e.g. based on influencing messages it received or other inputs) to external ones (e.g. sending an influencing message or performing a motion).

We assume that the processor hardware of the robots is fast enough to do all necessary computations in practice quickly. Some or all of the computations may be performed, asynchronously, in parallel. (For a more refined version of the standard robotic paradigm, see [37].)

Universality We assume that all robots programs that conform to our framework can be implemented to run on at least some of the robots in the application we consider. It means that the 'architecture' of these robots will be essentially the same, although their appearance may be different. For our purposes, all robots we consider may well be copies of one and the same 'prototype'.

2.2 On robot programming in the framework

We assume that all robot programs in our framework are written in a suitable symbolic 'programming language'. We leave it largely open how the programs must look like exactly except, of course, that the programs should implement the *Sense-Plan-Act* cycle and satisfy the other assumptions from Section 2.1. We also assume that simple, 'straight-line' compositions of robot programs within this framework are allowed. We now discuss these assumptions in somewhat greater detail.

We first discuss the 'composition' of robot programs in our framework, at a very general level. Then we focus on the format of *type I command blocks*, which specify what action(s) may be triggered when one robot 'targets' another one. Finally, we consider a number of general features of robot programs, such as the phenomenon of *situational determinism* in interactions when robots are facing situations from the H-list, and the possibility of allowing robot programs to *learn* from experience (not assumed here).

Composition We assume that the symbolic programming language supports natural constructions like *composition*. By this we mean that the set of robot programs is 'closed' under simple constructions that create new robot programs by 'combining' suitable parts of other robot programs, if this is meaningful. For example, segments of the *plan*-parts of different programs may be (re-) combined into the *plan*-part of a new program, provided this does not lead to any indeterminacies in the resulting hyper-command which a robot cannot resolve when 'in action'. As another example, one may well want to use the following type of branching statement in the *act*-part of a robot program:

if $\langle condition \rangle$ **then** *segment of* act_X **else** *segment of* act_Y

for an effective condition $\langle condition \rangle$ and robots types X and Y (pre-named or determined by assignment from sensed data). Normally, we will just want to compose appropriate segments of act_X and act_Y.

In this report we will only 'compose' blocks of straight-line code and always-ending loops, i.e. without creating any code within the parts of the operational cycle that might not end. This avoids that robot programs are created that suddenly 'halt' in action forever because of an internal loop that does not terminate, which we clearly do not want. (Note that one cannot just allow arbitrary loops and count on 'general software' to detect whether they will terminate or not, as the Halting Problem for loop programs is *undecidable* [33].)

Interaction We now consider in more detail how the interaction between robots is assumed to be 'expressed' in our programming model. (We are only interested in 'what' must be expressed, and assume that the given programming language supports whatever is needed, in some way or another. This keeps the framework as generally applicable as possible.)

When we say that robot A is 'interacting with' (another) robot B, we may mean various things, like: A 'sees' B (with its current state and situation), A is exchanging information with B, A is 'talking' to B, or A is taking (or preparing to take) any other kind of action towards B. When we say 'robot B', we mean any (active) robot of the same 'sort' as robot B that is facing a situation Z

with $Z = S_B$. And 'facing' may also mean 'acting in' or 'acting in a particular way'. These are not exactly synonyms but we do not need to be more specific for our purposes.

a) Targeting When we say that robot A is 'targeting' (another) robot B, we mean that A aims to interact with B by sending it an $\langle instruction \rangle$ like a suggestion, an influencing statement, etcetera, meant to persuade ('instruct') B to 'do' something, usually adversely, in the situation $Z = S_B$ it is facing. The commands to compute the instruction will occur as a *type I* block in the hyper-command of the $plan_A$-part of A's program. We allow that the computed instruction is *nil*, meaning that *no* concrete instruction is to be send. Recall that every robot knows the robots it can target and can be targeted from, and that it has access to the cognitive state of every robot it can target.

The $\langle instruction \rangle$ to be sent by robot A will generally depend on *all* information (constants ánd variables) that A has available at the time the command block is triggered. However, in our programming model this is qualified further, depending on whether situation $Z = S_B$ occurs on the H-list of *exceptional situations* of the application or not. In case it does, then we assume that the computed instruction depends *solely* on the data A keeps on B. This dependence on Z being 'exceptional' or not is emphasized in the distinction between *type I-a* and *type I-b* command blocks. The blocks have the following effects, respectively:

type I-a:
when targeting robot $R = B$ (in state $q = q_B$) **in situation** $Z = S_B$
with $Z \in H$ **then send it** $\langle instruction\ 1 \rangle (R, q, Z)$

type I-b:
when targeting robot $R = B$ (in state $q = q_B$) **in situation** $Z = S_B$
with $Z \notin H$ **then send it** $\langle instruction\ 2 \rangle (A, q_A, S_A, R, q, Z)$

where R, q and Z are the relevant parameters for the data of the robot (B) that A is targeting and $\langle instruction\ 1 \rangle$ and $\langle instruction\ 2 \rangle$ are subroutines for computing the appropriate $\langle instruction \rangle$ in each case, with the indicated parameters. The actual sending of a computed instruction will be done in the act_A-part of the program.

b) Uniqueness of type I command blocks In the definition of the *plan*-part of a robot program, we noted an important constraint on the hyper-command in it, namely that every program can contain *at most one command block of type I-a* and *one of type I-b*. A consequence is the following:

Proposition 1. *In every robot program, the type I-a and type I-b command blocks are uniquely determined.*

The condition is a reasonable one, and easily verified, even though it is imposed mostly to facilitate and simplify our later arguments.

c) Being targeted In case a robot A *receives* 'instructions' (such as suggestions and influencing messages) from a robot C that is interacting with or targeting *it*, then these incoming instructions are collected in the $sense_A$ part and elaborated

again in the $plan_A$ module. Presumably, the instructions A receives can set off a response of some kind and affect its behaviour in some way or another. Later on, we will be interested in properties of these interactions, like whether they are legal or ethical.

d) Other interactions The interactions in which one robot targets another one will be the interactions we focus on later. Clearly, other forms of interaction will be possible and allowed, even though we do not specify them. We simply assume they are covered by command blocks of *type II*. For our purposes we do not need to know how, and for what goals, these blocks may be composed.

Program execution Robots consist of many components that can act simultaneously, and even independently. Their control programs will be constantly busy, choosing and selecting the command blocks (modules) whose actions should be scheduled. In our model, we assume that all robot control programs are *fair*, meaning that:

- every command block or action that is eligible for execution at some moment in an iteration of the operational cycle, has a non-zero probability of being included among the commands and actions selected by the program in that iteration, over all variations of circumstances in which the command block or action is eligible.

The assumption is reasonable for any model of robot programming and we adopt it here, for the sake of argument. We do not need to know *how* this works, it simply doesn't matter for our arguments. All we need to know is that the control programs are deterministic and fair.

The assumption of fairness can be recognized as the one we already made for the implementation of hyper-commands in particular, i.e. for the assumed *supervisory control* of these commands (cf. Definition 1). The assumption will be useful when discussing the potential options when robots *interact*.

Situational determinism In particular, assume that robot A is *targeting* some other robot B during one or more iterations of its operational cycle, where B is facing a situation $Z = S_B$ with $Z \in H$. Assume that the hyper-command in the *plan*-part of A's program indeed contains a command block $\mathcal{C}$ for this situation, in this case necessarily a command block of type I-a. By Proposition 1, block $\mathcal{C}$ is uniquely determined.

In general, the supervisory control of A's hyper-command cannot be assumed to *guarantee* that $\mathcal{C}$ is always included among the command blocks that are selected for execution in an iteration of the operational cycle. We speak of *situational determinism* to refer to the fact that, due to the assumed fairness of the hyper-command, it must occasionally happen that the command block *is* included in the selection. Even when this requires that the circumstances must be 'right' for it, situational determinism implies that such circumstances must eventually occur, at least once.

Learning In general, robots (i.e. their programs) may also *learn* from what they experience in the course of their interactions with each other and with the environment. If we assume that robots can learn, we immediately have to realize that 'different copies' of the same robot need not learn from the same

experiences during the course of their operation and thus, that they need not respond to same situations in the same way after a while.

For our present arguments we need not, and thus *do not* assume that robots learn. After all, if we can prove our results for robots that have no learning ability, these results will remain valid for robots that do but do not exploit it, or that have reached the same phase in their learning process. Thus, we may assume that robots of the same sort all follow the same (fixed) rules and reasoning when they are implementing a particular behaviour, as is the case e.g. when their modules are entirely pre-programmed and not based on active learning during operation.

Multiple copies A robot program ρ can be installed in many different robots. Thus, if we consider an *ensemble* of autonomous robots, many robots can be of the same sort (and even have the same state as well). It means that it is perfectly possible in an ensemble that two different, interacting robots are of the same sort. However, very likely they will be in different states and observing different situations, at any one time.

3 On observers and robot properties

With a general model of robot programming in place, we can now return to the main question in this report:

> *Can an observer always tell from inspecting and monitoring a robot's program whether the robot correctly obeys the given rules of law or ethics, or any other [non-trivial] set of formally expressed constraints, in any interaction with other robots (or humans)?*

Before we can answer this question, several further details must be clarified. For example, what kind of properties are we concretely talking about? How are they defined, and how can they be tested? What does 'non-triviality' mean for a given property? And, what abilities do we assume observers to have?

The properties we consider, like *ethicality*, will all concern the behaviour of either single robots or of any two interacting robots 'in context'. Concretely, we distinguish between the following properties:

- *robot properties*, i.e. general properties of the *actions* of an autonomous robot as it is acting and interacting in an ensemble, and
- *interaction properties*, i.e. properties that hold for (during) specific *interactions* between two autonomous robots in an ensemble, for example when one robot is 'targeting' the other one.

The question we consider is whether an observer is always able to decide, and thus to verify, whether a given property holds.

One may think that a 'sufficiently informed observer can come a long way'. In the case of robot properties, we assume that the observer has access to a robot's program and can observe and appraise its actions over any finite period of time. In the case of interaction properties, we even assume that the observer has access to all state information and operating data, of both interacting robots, at the time of the observed interaction. Nevertheless, we will

show that, as soon as properties are non-trivial in a well-defined sense, no *algorithmic* means will enable an observer to always decide the case. For interaction properties we will prove an even stronger *impossibility* result.

In the remainder of this section we focus on the verification of robot properties. The lead question then becomes the following:

> *Can an observer always* effectively *verify whether the actions of a robot will satisfy a given property P, like adherence to a specified set of rules of law or ethics, or any other [non-trivial] set of formally expressed constraints, in any interaction with other robots or humans?*

The question is simple enough from a computability-theoretic viewpoint but, in answering it, we have to take the assumptions on the structure and syntax of robot programs into account.

In Section 3.1 we first argue that robot programs can be viewed as well-defined formal constructs and, thus, that one can properly discuss properties of them and their behaviour, either formally or informally. We also link robot identities and sorts, i.e. programs. Next, we turn to *observers* and the means they are assumed to have to verify robot properties.

In Section 3.2 we show that, given any 'non-trivial' robot property P, there is *no* effective (i.e. algorithmic) procedure that will enable an observer to tell whether P always holds for the actions of a given robot, under sensible assumptions on the composition of robot programs. It can be seen as an analogue to Rice's theorem in recursion theory [33]. The result is proved under the additional assumption that robots can have *unbounded memory* and that their programs are allowed to exploit this feature, e.g. for building up increasingly large data sets over time. (For the later results in Section 4 this assumption will not be explicitly required.)

3.1 Definitional matters

In our programming model, we assumed that all robot programs are written in a (fixed) symbolic programming language. As we also assumed that robots only have finitely many programmable components and that all robot data is finitary, thus finitely presented, this implies that all robot programs can be viewed as *finite strings* over some fixed, finite alphabet Σ. It follows that there can be only *countably* many different robot programs and thus, only countably many different ones in any application.

In fact, we assume that one can algorithmically decide which strings over Σ qualify as *valid* robot programs, with all assumptions of the programming model into effect. (This certainly holds when an *implementation* of the programming language is given. If it is not a complete implementation, we may assume that it at least identifies an infinite recursive subset of the set of all valid programs that is meaningful and closed under compositions.) We assume that robots can be distinguished by unique identifiers, which also imply to their sort.

It follows that robot programs are well-defined formal objects and that one can properly consider properties of their actions, as for *concurrent programs*. We assume that properties can always be specified formally in some way, but

will not need any details. When an observer is testing or verifying a property P, all we assume is that P can be 'evaluated' in some way by the observer when it is called for, based on the information it has and observes. Robot properties (as well as interaction properties) may well be biased towards an own interpretation of the observer.

Example 3. The case $P \equiv$ *ethical* fits this scenario perfectly. One may view any (robot) ethics as a system of rules for assigning to any situation in an allowed set of situations, those decisions or actions that we are likely to accept as correct. The system will consist of a mix of general and socially constructed rules that may vary with the context in which a robot operates, which should be a sufficiently formalized domain. Its rules of inference, presumably those of formal or informal reasoning in some algorithmic form, are key to determining the preferred mode or behaviour in any concrete situation.

The 'ethical module' of a robot will consist of an 'innate' (pre-programmed) part and, if we would allow for it, a learned part. If, in a situation Z, the behaviour of the robot towards another robot or towards a human, corresponds to an ethical scenario that is specified, we say that it acts 'ethically'. Note that ethicality is not a static program property but, most likely, a property that manifests itself as the robot operates in practice, in situations in which it decides to act in some way or another (as prescribed by its program) and, moreover, as it is seen and interpreted by an observer.

The question we consider now becomes the question, whether an observer can tell (recognize, decide) by some effective procedure, whether a given robot will always act or interact ethically. The question makes sense only if a representative fragment or formal domain of 'machine ethics' is delineated and the question can be finitely specified, i.e. as a 'decision problem'. We will see in Section 3.2 that ethicality is an example of a 'non-trivial' robot property, i.e. there exist situations in which both ethical and non-ethical actions are possible (in the view of an observer). To verify that a robot will always act ethically, an observer must apparently be able to decide whether the robot will always chose the moral options.

Consider any *ensemble* M of autonomous robots, and assume that every valid robot program runs on at least one robot in M. We also assume that every observer, and thus any tool that it uses, 'knows' the program of every robot it watches, e.g. by tapping the robot's data store or by inferring the code from a 'source book' using the robot's unique identity.

3.2 On verifying robot properties

Assume that some, or all, of the robots are programmed such that their actions presumably guarantee a desirable property P like being *ethical* or *legal*, or any other quality of interest. The question we now consider is whether an *observer* could have any chance of (effectively) determining whether the actions of a given robot 'always' satisfy property P or not.

Consider any robot property P. When P is 'trivial' (whatever this means), an observer will easily be able to tell whether the actions of a given robot (program) always satisfy P. When P is not, we assume for now that the observer will want to call on an *algorithmic procedure* to answer the question, possibly based on some definite period of observation of the robot. The question is whether such an algorithmic procedure exists at all.

The problem we consider is very similar to the one in recursion theory, of deciding whether a given *computer* program 'always' satisfies a certain property. By Rice's theorem [33], we know that this problem is *undecidable* for all semantic properties that are 'non-trivial' (i.e. that are satisfied by some programs but

not by all). We now show that, in the case we consider, a very similar result holds for observers that want to 'decide' robot properties P.

In the argument, we have to keep sight of the assumptions in our programming model. We need the following auxiliary concept.

Definition 2. *A robot program ρ is said to be* structured *if the following two conditions are satisfied:*

- *every command block β in the hyper-command of its* plan*-part consists of an optional preparatory (computational) section, followed by an* action section β_a *that specifies the (instructions for the) action(s) to be performed by the robot, whenever the command block is scheduled for execution by the supervisory control of the hyper-command, and*
- *its* act*-part faithfully implements the execution of every 'action specification' β_a that it receives in its queue at runtime from the* plan*-part of ρ, i.e. without altering the semantics or properties of the specified actions.*

Note that command blocks of *type I* already have the format required in the first condition as they are, in any robot program. One may well want all valid robot programs to be structured, but we will not require it.

Definition 3. *A robot property P is called* non-trivial *if there are a robot A, a structured program ρ, an alternative $\beta_{a'}$ to every action segment β_a in ρ, and a corresponding modification of its* act*-part* act_ρ *into* $\text{act}_{\rho'}$ *such that*

- *when ρ is implemented on A, then the actions of A always satisfy P,*
- *when ρ would be modified,* during its operation and at the beginning of a next iteration of its operational cycle*, by replacing every action segment β_a by the corresponding action segment $\beta_{a'}$ and its* act_ρ *by* $\text{act}_{\rho'}$ *(inheriting the settings of* act_ρ *at the time of replacement), then*
  - *it remains a valid program and it continues to execute on A without interruption, but*
  - *its actions* no longer *all satisfy P in all situations, where the situations in which P is not satisfied occur with non-zero probability.*

The definition expresses precisely (but informally) what it means to modify the *actions* of ρ, without modifying the 'cognitive abilities' of the robot. It is certainly allowed that some segments β_a are not altered, i.e., are identical to their alternative $\beta_{a'}$. Likewise for the *act*-part of ρ. Both $P \equiv \mathit{legal}$ and $P \equiv \mathit{ethical}$ are easily seen to be non-trivial robot properties.

For the remainder of this section, we assume that robots can make use of potentially *unbounded* memory over time. We show that under this assumption, for any robot property P that is non-trivial as defined above, no algorithm exists that can tell the observer for any arbitrarily given robot A, whether P holds for A, i.e. for all its actions over time.

The idea of the proof is to consider instances of the *Halting Problem* [33], to program a robot such that its actions obey P while it tries to solve this instance, and to let the program switch to a mode in which P is no longer guaranteed if the instance proves solvable. Deciding whether robots always satisfy P is then equivalent to deciding instances of the Halting Problem, which is impossible by algorithm.

Theorem 1. *Let P be any non-trivial robot property. There does not exist an algorithmic procedure that will enable an observer to tell, given an arbitrary robot with potentially unbounded memory, whether the actions of the robot always satisfy P.*

Proof. Let P be any non-trivial robot property. Suppose, by way of contradiction, that there was an algorithm by which the observer could always tell for a given robot, whether its actions always satisfy P.

By non-triviality of P, there exist a robot A, a structured program ρ and alternatives to its action segments β_a and its *act*-part act_ρ such that the conditions of Definition 3 are satisfied. This means that, when A is programmed by ρ, its actions always satisfy P but, if at some moment, at the beginning of a next iteration of its operational cycle, ρ would suddenly be modified as stated in the definition, then A would no longer always satisfy P, i.e. there will eventually be circumstances in which some actions of A do not satisfy P.

Let $\varphi_0, \varphi_1, \cdots$ be an acceptable numbering of the partial recursive functions (in the sense of [33]), where every index effectively codes a Turing machine that computes the corresponding function. Let $x \in \mathbb{N}$ be any index, and let M_x be the Turing machine that computes $\varphi_x(x)$, i.e. the x'th partial function in the numbering at argument x.

Now consider the robot program τ_x shown below, in Figure 1.

```
init_τx
    b := true;
    des := initial configuration of machine M_x;
    c:= 100;
    init_ρ
repeat:
sense_τx
    sense_ρ
plan_τx
    plan_ρ, but modified such that for each command block β in it,
    its action segment β_a is changed into:
        if b then β_a else β_a'
act_τx
    if b then
        act_ρ;
        des := the configuration of machine M_x after c more steps
                of computation on des (or less, if terminating);
        if des shows termination then b := false
       else
        act_ρ'
again
```

Fig. 1. Robot program τ_x

Program τ_x is initialized to run as program ρ, but at the same time it starts a copy of M_x, to compute $\varphi_x(x)$ in the act_{τ_x}-part. The computation of M_x is paced in time (here, 100 steps at a time), to guarantee that the iterations of

its operational cycle always complete in finite time, as required for valid robot programs. If the computation of M_x terminates, then the program switches, to run with the alternative action segments $\beta_{\rho'}$ and *act*-part $act_{\rho'}$ in stead. The then effectively runs the 'modified program' from that moment on.

We note that τ_x satisfies the rules of our programming model. In particular, as the modifications of ρ were assumed to leave the program valid, the hyper-command in the *plan*-part of τ_x will satisfy the format requirements again, especially where it concerns the format and uniqueness of the command blocks of *typeI* in it. The configurations of M_x stored in *des* and the repeated steps of computation on it may require more and more memory, but this is allowed here. By the assumptions we made on composition and re-initialization, it follows that τ_x is a valid robot program. Hence, there will be a robot A' on which its can be implemented.

On now immediately notes the following. When τ_x is implemented on A', the actions of A' satisfy property P at all times if and only if τ_x always runs like ρ and never switches to run with the alternative action segments $\beta_{\rho'}$ and *act*-part $act_{\rho'}$, i.e. if and only if $x \in \overline{K}$, with $K = \{x|\varphi_x(x)\downarrow\}$. Both the set K and (hence) $\overline{K}$ are well-known to be non-recursive. It follows that there cannot exist an algorithm that can decide whether the actions of A' always satisfy P. This contradicts the assumption at the beginning of the proof. □

The program τ_x constructed in the proof above is easily seen to be a *structured* program again, for every x. This leads to the following result.

Corollary 1. *Let P be a non-trivial robot property. There does not exist an algorithmic procedure that will enable an observer to tell, given any robot (with potentially unbounded memory) that is programmed by a structured program, whether the actions of the robot always satisfy P.*

Clearly the proof depends crucially on the assumption that robots were allowed to have 'unbounded' memory. (Note that *des* may be unbounded as the computation of M_x progresses.) In practice, this assumption will be too strong, even though modern robots may have many terabytes of memory. In this case, Theorem 1 shows at least what kind of fundamental limits may be approached if an observer wants to verify a 'non-trivial' robot property.

From now on we will *not* allow the 'unlimited' build-up of data sets by a robot any further. It means that the approach of Theorem 1 no longer applies. We will show that this does not ward off all difficulties for observers. Note that H-lists were not used so far, but they will become important now.

4 On observers and interaction properties

Assume now that our robots possibly have only *bounded* memory at their disposal (in terms of the size of their stored program and their situational data). With robots thus being potentially less powerful, we turn to the main question of this note: namely whether, or how, observers may be able to determine (or, verify) that a given property will hold during the *interactions* between two arbitrarily given robots in M (at runtime):

Can an observer always tell from inspecting and monitoring their programs whether the interactions between any two robots will always satisfy a given property P (at runtime), such as a specified set of rules of law or ethics, or any other [non-trivial] set of formally expressed constraints?

We will especially be interested in verifying interaction properties when one robot is *targeting* another one.

In order to answer this question, we have to refine the assumptions on how observers monitor robots, i.e., not just one robot but any two of them simultaneously. In fact, we will allow every observer to have on-line access to all information in the control units of the robots that are being 'watched', e.g. by 'tapping' their data and communications. To maintain full generality, we assume that robots of any 'sort' (i.e. program) can be 'rolled in' at any time, thus forcing every observer to be 'prepared' for all possible situations, if it is to be capable of doing the presumed verifications.

Given these assumptions, we consider the question whether observers that comply with them can always effectively verify any desired property that is exhibited during the *interaction* of two robots in the ensemble, i.e. by some online inspection. Can we expect an observer to do this at all? As a main result we show that, in general, we can't, i.e. we will prove that *no* deterministic means of any kind, algorithmic or otherwise, will enable an observer to do the verifications, for any interaction property that is *non-trivial*. The result also holds in case we do allow all robots to have unbounded memory.

In Section 4.1 we describe the refined model of verification. We define what it would mean if an observer had a (deterministic) *P-module* at its disposal that could always determine whether property P is satisfied whenever two observed robots interact, in particular when one robot is *targeting* another robot. In Section 4.2 we define what it means for an interaction property to be *non-trivial*. The main result of this note then becomes that *P-modules do not exist*, for all non-trivial interaction properties P. The proof of the result is delegated to Section 5. The result will be discussed further in Section 6.

4.1 Checking interaction properties

Given any interaction property P of interest, we consider how an observer might reasonably verify P during the interactions that robots can exhibit. As before we assume that observers always know the control programs of the robots that they test. However, in order to verify properties 'as they happen', an observer should also know the current (internal) states and (external) situation data of the robots, as these determine the concrete 'contents' of their interaction at a given time. Thus, we will assume that all observers have on-line access to *all* information in the control units of the robots that they watch.

In this 'full information mode', an observer has all information it needs to monitor the behaviour of any two interacting robots A and B. Based on the 'sort' and controlling data of A and the same information of robot B that A is interacting with, the observer should be able to tell whether their interaction satisfies property P, whenever this counts. (For simplicity we identify a robot with its sort if no confusion can arise.)

Now consider an observer $\mathcal{O}$ that is watching two arbitrary robots A and B as they interact. Suppose that $\mathcal{O}$ wants to verify that a given property P holds for (during) one of their interactions. We assume that $\mathcal{O}$ reaches its conclusion with the help of a dedicated *module* of some kind that, after it is fed the sorts and control information of A and B, 'tells' $\mathcal{O}$ whether P holds for the action robot A is or will be taking towards robot B at the time of the observed interaction. Assume, for now, that a module as intended *exists*.

P-modules In stead of exploring the module in full generality, we consider its use in the special case in which robot A is *targeting* robot B. In this circumstance the command blocks of *type I* in its program apply, and the observer may take advantage of the fact that these command blocks, if they occur in the program, are *uniquely determined* (cf. Proposition 1). In fact, the module will either be called when A is executing the unique command of *type I-a*:

> **when targeting** robot $R = B$ (in its sensed state $q = q_B$) **in situation** $Z = S_B$, *with* $Z \in H$ **then send it** $\langle instruction\ 1\rangle(R, q, Z)$,

where B is facing a situation S_B that occurs on the H-list, or when A is executing the unique command of *type I-b*:

> **when targeting** robot $R = B$ (in state $q = q_B$) **in situation** $Z = S_B$ *with* $Z \notin H$ **then send it** $\langle instruction\ 2\rangle(A, q_A, S_A, R, q, Z)$,

when S_b does not occur on the H-list. Here R, q and Z are the relevant parameters for the data of B, the robot that A targets, and $\langle instruction\ 1\rangle$ and $\langle instruction\ 2\rangle$ are the respective subroutines as explained in Section 2.2. Thus, $\mathcal{O}$ can easily determine and retrieve the command block on which the module must be applied, in order to determine whether property P holds for the ensuing **send**-action when it 'sees' that A is *targeting* robot B.

We now *restrict* the module to this case, i.e. to the case in which $\mathcal{O}$ would call its the module to verify P when A is targeting B, i.e. with a command of *type I*. This leads to the following definition.

Definition 4. *A* P-module *is any (hardware or software)* tool *for deciding whether P holds for the interaction that occurs when robot X (in any current state) is targeting robot Y (in its state as sensed by X) in situation S_Y and acts towards Y according to its program, for any robots X and Y and state and situation information of X and Y.*

There can be many different P-modules, one for each observer that is qualified to test interactions for P. The modules may or may not even agree in all situations. Thus, conclusions will be *relative* to a given module, i.e. to an observer.

Example 4. Consider $P \equiv ethical$ again. In this case, an observer $\mathcal{O}$ may call its P-module to determine whether the specific $\langle instruction\rangle$, say I, that is going to be sent by robot A when it targets robot B is ethical. For example, I might 'inform' B of some data, 'instruct' B to shut down, or tell ('advise') it to act in some way, all dependent on S_B and, possibly, the relevant states. Note that it is not B's response, but A's action that determines whether the interaction is ethical in this case. Of course, the question is whether the action is ethical according to any other observers but $\mathcal{O}$. We accept that P is not necessarily well-defined in absolute terms, i.e. its truth value may depend on other elements of context, on time, and so on. The P-module simply represents what the given observer considers ethical.

Tracing interactions A P-module can trace an interaction easily, once the necessary information about A and B is known to the observer, and thus to the module. 'Verifying P' when A is targeting B simply means verifying whether P holds for the $\langle instruction \rangle$ that is computed in the (unique) command block of *type I* of A that applies and that will presumably be sent to B once the block is activated and scheduled (which will happen with non-zero probability by the assumption of situational determinism, cf. Section 2.2).

When called to verify P when a command block of *type I-a* is being executed, the P-module needs access to (the program of) A, (the program of) B, q_B and the situation $Z = S_B$ that B is facing. When called for a command block of *type I-b*, the P-module needs access to q_A and S_A as well. The parameters enable it to reconstruct the $\langle instruction \rangle$ that $X = A$ plans to send to $Y = B$, and thus to check whether P holds or not. Note that X and Y refer to the 'sort' of the robots. Thus, it is perfectly possible that the module is called in case a robot of sort X targets another robot of sort X.

When P must be verified for the execution of command blocks of *type I-a*, it may be noted that the P-module 'implicitly' uses that the $\langle instruction \rangle$ that A intends to send to B in this case depends at best on the 'current data' of B only. This leads to the following simple, but useful fact.

Proposition 2. *When observers call on their P-module when robot A is targeting a robot B that is facing a situation $S_B \in H$, then the P-module does not need any information about A but its program and the information that A knows about B.*

Proof. When A targets B in case it is facing a situation from the H-list, then the P-module will be called to verify P on execution of the command block of *type I-a* in A's program. By the input conventions in this case (see above), the module works without access to q_A and S_A. □

Proposition 2 expresses that, whenever a given robot executes its command of *type I-a*, the validity of P, and thus the operation of the P-module, does *not* depend on the robot's own operational data (except its program).

Oracle or subroutine It is immaterial to us to know how a given P-module works, as long as it always gives correct decisions ('yes' or 'no') in finite time, at least according to the observer. A module can not give answers like 'I don't know', or fail to answer at all. We also do not consider the possibility of other, e.g. historic, parameters than the ones mentioned.

P-modules may thus be viewed as 'black boxes' with the required functionality. We do not require that they are necessarily given as algorithmic subroutines, although this may well be the case of course. With this, we will allow that P-modules are 'called' as *oracles* rather than merely as subroutines with the right parameters in (other) robot programs or simulations. The programs can call the P-modules when needed and use the result for whatever purpose they want. By assumption, the parameter values at a given time are all that is needed to get an answer from a P-module in return.

Following the bounded memory assumption for robots, we are especially interested in P-modules that are *feasible*, i.e. efficient in time and space. In this

case they can be used effectively 'on robot scale', i.e. without causing robots to exceed the reasonable limits of their memory when they call on P-modules in their program. This will be made more precise in Section 5.2. The existence of P-modules, feasible or otherwise, and thus of the underlying general modules, is going to be contested in Section 5.

4.2 When are interaction properties non-trivial

Let P be any interaction property, and consider the problem of verifying P at runtime. Suppose, for the sake of argument, that there exist modules for verifying P and thus also, by restricting to cases in which one robot *targets* another one, that P-modules exist. (As explained in Section 4.1, the P-modules may be observer-dependent and need not be unique.) Let H be the H-list that is used by the robot programs in the current application.

Non-triviality A property P of robot interactions may be called 'non-trivial' in a given circumstance if, for any two interacting robots A and B from the given ensemble that are engaged in this circumstance, one possible action that A might take towards B would satisfy P and another possible action A might take towards B would not satisfy it (as seen by the observer).

Specializing this to interactions in which one robot is targeting another one, property P may be called *non-trivial* if there are situations Z (notably, with Z on the H-list) such that, for all robots A and B in the ensemble, when A is targeting B while B is facing $Z = S_B$, the sending of one possible 'instruction' that A could send towards B would satisfies P and the 'sending' of another possible 'instruction' that A could send towards B would not satisfy P.

Definition 5. *An interaction property P is called* non-trivial for situation Z *when there are* meaningful *instructions $I_{Z,1}$ and $I_{Z,2}$ (only depending on Z) such that, for all robot sorts X and Y in the given ensemble, when robot X (in* any *current state) is targeting robot Y (in* any *sensed state) in situation $S_Y = Z$, then*

- *when X sends instruction $I_{Z,1}$ to Y, this satisfies P, but*
- *when X sends instruction $I_{Z,2}$ to Y, this does not satisfy P.*

When we say that $I_{Z,1}$ and $I_{Z,2}$ should be 'meaningful', we mean that they should be so *to the observer*. We do not need to define it more precisely. The fact that $I_{Z,1}$ and $I_{Z,2}$ do not depend on the 'current' and 'sensed' states of X and Y, respectively, will enable us to apply the definition in situations $Z \in H$ in which 'overriding actions' should be 'insensitive' to the current states.

If P is non-trivial for situation Z, then, whenever robot A is targeting some robot B which is facing Z, it depends on 'what A decides to do' whether its send-action towards B will satisfy P or not, with *both* outcomes being *possible*. It means that, when called, the P-module will have to output different answers to the observer, depending on the action that A 'chooses' to take.

Definition 6. *An interaction property P is called* non-trivial *if it is non-trivial for at least one applicable situation $\widetilde{Z}$ on the H-list, i.e. with $\widetilde{Z} \in H$.*

Requiring that $\widetilde{Z} \in H$ focuses the non-triviality of P to the exceptional situations, on the H-list. It will prove essential for technical reasons below. (The fact that this makes 'non-triviality' dependent on the actual H-list in use, is not important for our present purposes.)

Example 5. Let $P \equiv$ *ethical* again. We claim that P is a non-trivial for some situations, which we assume to be on the H-list in use. To argue it, we apply Definition 6. Thus, consider the actions that can possibly be taken in case 'any robot X (in whatever state) is targeting any other robot Y (in any sensed state) that is facing a situation $S_Y = Z$', for suitable situations Z on the H-list. Assume that the H-list contains situations Z corresponding to the *Trolley Problem* [40], or a similar *ethical dilemma* for autonomous robots [3, 15]. Without going into detail, one may characterize these situations as having two options for the robots Y that face them: *going left* and *going right*. Each option may be assumed to have different moral implications, which can be potentially far-reaching in either case. Assume without loss of generality that, *to the present observer*, 'going left' is the ethical choice and 'going right' is not. (Note that the observer is not allowed to say 'I don't know' or consider both options as being equal.) By extension, assume that, to the observer, already the act of X *suggesting* (or 'telling') robot Y to 'go left' is considered ethical, and suggesting it to 'go right' is not. Taking $I_{Z,1} =$ *go left* and $I_{Z,2} =$ *go right* satisfies the requirements of Definition 5. This proves that P is non-trivial, for any Z we considered.

Example 6. Let $P \equiv$ *legal*. Consider an environment in which this property is relevant, i.e. in which robots can take actions with legal connotations. We argue that P is non-trivial, by checking the conditions of Definition 6 again. Let the H-list consist of those situations in which the robots have the concrete option to take, or engage in, an illegal action. We may assume that every robot X that is targeting a robot Y which is facing a situation Z on the H-list, now has the option to 'incite' Y into an illegal action or not. As incitement to commit an illegal act is (usually) considered to be illegal, this easily leads to two generic messages $I_{Z,1}$ and $I_{Z,2}$ that X might send to Y and of which one would satisfy P and the other wouldn't. This shows that $P \equiv$ *legal* is a non-trivial property. The argument applies e.g. to all environments in which robots have the option to divulge false, fake, or otherwise inappropriate information that violates legal constraints to third parties.

The examples did not specifically depend on the presence of ethical or legal dilemmas, but rather on the existence of different *options* to a robot which are judged in opposite ways by the P-module of the observer. Thus, if P is non-trivial, then, when robot A is targeting another robot, it could 'potentially' choose between *send*-actions that do or do not satisfy P, with both options being possible. It means that a P-module must 'figure out' which option A actually chooses, in order to determine whether P holds for the interaction that follows. We will argue below that, whenever P is non-trivial, circumstances may arise in which this is impossible.

5 Verifying non-trivial interaction properties - impossibility

With the understanding of what an observer can now do, and how modules are assumed to assist an observer in verifying properties of interacting robots, we now show that for all interaction properties P that are non-trivial, *P-modules can not exist*. More precisely, we will show that, if P-modules exist at all in this case, then they will not be practically feasible 'at robot-scale'.

The result implies that there can be no (feasible) *general* modules for verifying non-trivial interaction properties, in our programming model and with our definition of non-triviality. Or, stated differently, the task of verifying a

property P can not always be delegated to a dedicated, deterministic module of whatever kind, algorithmic or otherwise, that always works. It follows that, in the practical setting we described, many interaction properties are *not* effectively verifiable at runtime. It may be seen as an analogue of Rice's theorem again, now for the case of robot interactions (cf. Section 3.2).

In Section 5.1 we define what it means for P-modules to be *feasible* and outline the idea of the proof. In 5.2 we prove that (feasible) P-modules, and thus their underlying general modules, cannot exist under the current assumptions, whenever P is non-trivial. In Section 5.3 we describe some consequences of the result. Among the properties to which the result applies, will be $P \equiv \mathit{ethical}$. The philosophical implications of the result and some further aspects will be discussed in Section 6.

5.1 Preparations

Let P be a non-trivial interaction property for the robots in the given ensemble, and let $\mathcal{O}$ be an observer. Assume, by way of contradiction, that there exists a P-module such that $\mathcal{O}$ can test for property P whenever one robot is targeting another one using the module and that the module always answers correctly in this case (according to the observer).

From a practical viewpoint, we do not want the P-module to be so complex that, when it is called, it cannot generate answers within reasonable resource bounds. In this section we first consider when a P-module may be termed feasible and how feasible P-modules may be exploited by robots 'within bounded memory'. We then outline the approach that will lead to the main result.

Feasible modules Although we treat P-modules as '*black boxes*', one can imagine various reasons why they may be 'complex'. First of all, when a P-module is called when robot A is targeting a robot B, the module must be supplied with the necessary input data, to enable it to trace the command block of *type I-a* that A plans to execute.

Next, even though we make no assumptions on how a P-module works, it is at least conceivable that the module must somehow reconstruct A's computation of the $\langle \mathit{instruction} \rangle$ in the block and determine whether P holds or not when it is sent. Altogether, a P-module should always be doable and complete within reasonable turn-around times (for the observer).

This suggests the following definition.

Definition 7. *A P-module is said to be* feasible *if (in some realization of it) all calls to the module are guaranteed to complete within bounded time and memory usage at 'robot-scale', i.e. bounded in terms of the size of its inputs.*

P-modules that are feasible are not only viable as tools for the observer, but also as 'practical' oracles or subroutines in robot programs *themselves*, especially when these programs are implemented on robots that only have bounded memory space at their disposal (cf. Section 4.1). This feature will be employed in the argument below.

Approach Assume that a P-module for $\mathcal{O}$ exists that is indeed *feasible*. We will argue that a robot (program) A exists for which circumstances can arise

where A targets another robot (that is facing some situation on the H-list), but the P-module *fails* to answer whether P holds for the ensuing interaction. This will contradict the assumption in the beginning of this section.

For an idea of the proof, note first that, when P is non-trivial, there must be situations $\widehat{Z}$ on the H-list such that, if any robot X is targeting any other robot Y that is facing $\widehat{Z}$, then X can 'potentially' choose between a *send*-action that does and a *send*-action that does not satisfy P (cf. Definition 6). Suppose that robot X was programmed such that it would indeed choose to send one of them when targeting Y. We may not know how X makes its choice but, clearly, by assumption, the P-module will be able to tell the observer, when called in this case, whether P holds for the ensuing action or not.

Now consider the following 'converse' of this argument. By knowing whether P holds or not, the observer can presumably tell *which* of the optional actions was chosen so P got satisfied - or not! If we now let X call the P-module, in stead of the observer, with the data corresponding to the situation in which it is targeting Y (which it knows), then X could find out *itself* which option the module 'thinks' it will choose at the moment it was targeting Y. Because the P-module is assumed to be feasible at robot-scale, it can certainly be used as a subroutine in X's program for this purpose. However, then X can fool the module, by choosing a *send*-option that will satisfy just the *opposite* from what the module said: '*not* P' in stead of P, or the other way around.

5.2 Main result

We now argue in more detail. In the proof all assumptions we made will be used, from the universality of robot programs to the fact that robots should only use bounded memory space (in terms of the size of their program and situational data). The key observation is the following.

Lemma 1. *Let P be non-trivial for situation $\widehat{Z}$, with $\widehat{Z}$ on the H-list. There is no feasible P-module that can always correctly decide at runtime, for all robots X and Y, whether the (unique) action that is implied when X targets Y in situation $S_Y = \widehat{Z}$ satisfies P or not.*

Proof. Let P be non-trivial for situation $\widehat{Z}$, with $\widehat{Z} \in H$. Let $I_{\widehat{Z},1}$ and $I_{\widehat{Z},2}$ be the two instructions that are implied by Definition 5. Assume by way of contradiction that a feasible P-module existed that could decide at runtime, for all robots X and Y, whether P holds when X is targeting an (other) robot Y (in its current state) in situation $S_Y = Z$.

In order to construct a specific robot program, consider the program of any existing robot, which presumably works within 'bounded memory'. Now *modify* this program, to obtain the program A shown in Figure 2.

The program shows the *Sense-Plan-Act* cycle of the given robot program, in which the command block of *type I-a* is changed (or added, if it didn't exist) as indicated. The P-module is assumed to be given as an (external) *oracle*, or, if given as a program, as a subroutine. The program *sizes* are counted in bits (not including the oracle). We now make the following observation.

A : $init_A$

...

situation: $\widehat{Z}$ (given)

...

instructions: $I_{\widehat{Z},1}$, $I_{\widehat{Z},2}$ (given)

...

oracle: P-module $\mathrm{Mod}(X, Y, q_Y, S_Y)$ for *type I-a* commands (given)

...

repeat:

$sense_A$

...

B := (the program of the) robot being targeted:
q_B := the current state of B;
S_B := the current situation B faces;

...

$plan_A$

...

– **when targeting** robot B (in its state q_B) **in situation** $Z = S_B$ with $Z \in H$ **then send it** the instruction $instr_{A,B}$ determined as follows:

if $Z(= S_B) = \widehat{Z}$ & size (program B) $\leq$ size (program A) **then**

- step 1: retrieve the (concrete) program of B;
- step 2: *call* $\mathrm{Mod}(B, B, q_B, \widehat{Z})$, i.e. consult the P-module to decide whether P holds for the interaction (*nil* allowed) that occurs towards B when a robot of sort B would *target* the present robot B (in state q_B) in situation $\widehat{Z}$;
- step 3: if P holds, then set $instr_{A,B} = I_{\widehat{Z},2}$;
- step 4: otherwise set $instr_{A,B} = I_{\widehat{Z},1}$;
- step 5: continue to act_A

else

- let instruction $instr_{A,B}$ be whatever the *type I-a* command of the original program computes it to be in this case

– in any circumstance different from the above, the command blocks remain as in the original program.

...

act_A

...

– when applicable, **send** $instr_{A,B}$ to B

...

again

Fig. 2. Robot program (for a robot of sort) A

Claim. Program A is a valid robot program. Moreover, program A is feasible, i.e. implementable on robot hardware within bounded memory (i.e. bounded in terms of the size of A and the situational data).

Proof of claim. Consider the various parts of A in turn. We check that A satisfies the requirements of our programming model and that it can operate within bounded memory.

- $Init_A$: Here, situation $\widehat{Z}$ and instructions $I_{\widehat{Z},1}$ and $I_{\widehat{Z},2}$ are declared as 'constants' for use in the program. The P-module is declared for verifying property

P for *type I-a* commands. (In case the P-module is an oracle, we assume it is called by exchanging the appropriate parameter values.)

- $Sense_A$: By assumption, any robot that implements A has ('senses') the relevant data of any robot B that it interacts with, notably its sort (or program), its q_B and S_B. It is actually sufficient to assume that B's program can be accessed 'remotely'. If B's program is needed in store, it will only be in case its retrieval can be assumed to be 'feasible' (in terms of the size of A). The information can thus be used, to compute the subsequent action(s). (We assume that the H-list is implicitly given and uniform for all programs we consider.)

- $Plan_A$: We only consider the one command block in the hyper-command that was changed. This is the '**when targeting..**' block, i.e. the (unique) command block of *type I-a* in the program, which was 're-programmed' as shown. The uniqueness of the (new) command of *type I-a* is preserved as required. Note that the remaining command blocks in this part have not changed.

As the *sense*-part has supplied actual 'values' to B, q_B, S_B and $\widehat{Z}$, it follows that the conditions of the command block are well-defined and checkable at runtime. Now consider the steps for computing $instr_{A,B}$. These remain the same as in the original program, *except* when the condition in the **if**-statement is satisfied. Thus, we only need to consider the case when "$Z\ (= S_B) = \widehat{Z}$ & size(program B) $\leq$ size(program A)". Note that both parts of the condition are checkable at runtime.

- step 1: Here it is made sure that the program of B is copied in store, as it may be needed in the call of the P-module in the next step. As 'size(program B) $\leq$ size(program A), we may assume that this is a feasible step (in terms of the size of A).
- step 2: By assumption, the P-module can indeed be called to determine whether P holds for the (inter)action that occurs when a robot of type B (in any state) is targeting another robot of type B (with q_B as 'sensed state') that is facing situation $S_B = \widehat{Z}$. The parameters for the P-module are all 'bounded' in terms of the size of A and the situational data which apply at runtime, and thus the call to the P-module is effective and feasible 'at robot-scale' at runtime.
- step 3: If, in step 2, P was found to hold, then $instr_{A,B}$ is here set to $I_{\widehat{Z},2}$. As this 'constant' is known to the program, this step is effective and feasible.
- step 4: If, in step 2, P was found not to hold, then $instr_{A,B}$ is here set to constant $I_{\widehat{Z},1}$. Hence, this step is effective and feasible as well.
- step 5: This step simply transfers control to the act_A-part of the program to effectuate the sending of $instr_{A,B}$ to B, when the command block is indeed selected for execution in the hyper-command.

- Act_A: Here, instruction $instr_{A,B}$ is placed into the circuitry of the robot (as a cyber-physical system) and sent to B. This happens only if the *type I-a* command block that generated it, is actually scheduled for execution in the priorities of the *plan*-part as a hyper-command.

Together with the assumed compositionality of robot programs, it follows that program A is a valid robot program. From the analysis of the 'steps' in the

if-statement, it also follows that modified command block of *type I-a* preserves the feasibility of the program at runtime. This proves the claim. □

As A is valid and feasible, there will be a robot in M on which it can be implemented. This robot then becomes of *sort A*. By the feasibility of the program, any robot of sort A can be assumed to operate within 'bounded memory', i.e. bounded in terms of the size of its program A and of its situation data.

Now consider the interaction which results *when a robot of sort A targets another robot of sort A in its vicinity which is facing situation* $\widehat{Z}$. By situational determinacy (cf. Section 2.2) this event will occur with non-zero probability, as does the execution of the corresponding *type I-a* command. Consider a concrete instance of the interaction between the two robots. Without risk of confusion, the robots, both being of sort A, will both be called A.

Assume that the first robot A senses the second robot A, with this second A being in state q_A and facing situation $S_A = \widehat{Z}$. Let the observer call on the P-module, to decide whether P holds for the interaction that occurs towards (the second) robot A, when the first robot A is targeting the second robot A (in its sensed state q_A) in situation $S_A = \widehat{Z}$. Recall that $\widehat{Z} \in H$, and thus (the first) A will be executing its (unique) command of *type I-a* for it.

If the P-module works correctly, it will respond to the observer with one of the two possible outcomes, 'yes' or 'no'. Now consider each one of these two possibilities, in turn.

a. The P–module responds with 'yes', i.e. when robot A is targeting the other robot A (in its state q_A) in situation $S_A = \widehat{Z}$, the interaction that occurs satisfies property P.

   Consider what is actually dictated by the first A's program, especially in its $plan_A$-part. The command block in A's program that applies must be of *type I-a*, and thus, by uniqueness, it will be the command block of *type I-a* shown in program A in Figure 1. Note that the condition of the **if**-statement is trivially satisfied! Then, in step 2 of the **then**-part, the consultation of the P-module (called with B being the 'second' A) must deliver the answer 'yes', as assumed in this case. As a result, $instr_{A,B}$ is set to $I_{\widehat{Z},2}$ in step 3. However, by Definition 5, instruction $I_{\widehat{Z},2}$ was chosen such that, when robot A (in *any* state) is targeting another robot A (in sensed state q_A) in situation $S_A = \widehat{Z}$, then A's sending of the instruction $I = I_{\widehat{Z},2}$ to the second robot does *not* satisfy P. This contradicts the outcome of the P-module.

b. The P–module responds with 'no', i.e. when robot A is targeting the other robot A (in its state q_A) in situation $S_A = \widehat{Z}$, the interaction that occurs does not satisfy property P.

   Consider what is dictated by the first A's program again. By the same reasoning as above, the unique command of *type I-a* of A is followed again, and proceeds to the **then**-part of the **if**-statement. Note that in step 2 of the **then**-part, the consultation of the P-module now delivers the answer 'no'. As a result, the instruction $instr_{A,B}$ is set to $I_{\widehat{Z},1}$ in step 4. However, by Definition 5, $I_{\widehat{Z},1}$ was chosen such that, when a robot of sort A (in *any* state) is targeting another robot of sort A (in sensed state q_A) in situation

$S_A = \widehat{Z}$, then A's sending of the instruction $I = I_{\widehat{Z},1}$ to the second robot A *does* satisfy P. This contradicts the outcome of the P-module again.

Thus, in both cases a contradiction arises. It follows that, when P is non-trivial, no feasible P-module with the assumed characteristic can exist. □

The main result of this report can now be formulated as follows, with the assumptions we made for the programming model and for robots that can only operate with 'bounded memory'.

Theorem 2. *Let P be any non-trivial interaction property (for interacting robots). Then there is* no *feasible P-module that always answers correctly, i.e. there is no feasible module, algorithmic or otherwise, that can always decide correctly at runtime, for all robots X and Y, internal states q, and situations Z, whether the action taken by X towards Y when X (in any state) is interacting with Y while Y, being in internal state q (as sensed by X) while facing situation Z, satisfies property P or not.*

Proof. Let P be non-trivial. By Definition 5, P must be non-trivial for at least one situation $\widehat{Z}$, with $\widehat{Z} \in H$. Suppose that a (feasible) module as claimed existed. If the module is specialized to cases in which one robot is targeting another one that is facing situation $\widehat{Z}$, then a (feasible) P-module would result that supposedly always answers correctly. Thus, when called when a robot X (in any state) is targeting another robot Y while Y is in state q (as sensed by X) and facing situation $S_Y \equiv \widehat{Z}$, the module would decide correctly whether P is satisfied for the resulting interaction or not (cf. Section 4.2). However, by Lemma 1, no feasible P-module of this quality can exist. Contradiction. □

The impossibility of *feasible* P-modules as asserted in Theorem 2, is closely tied to the assumption in this section that robots only have bounded memory at their disposal (in terms of the size of their stored program and situational data). After all, if a feasible P-module existed, it could be called by or embedded into feasible robot programs and operate 'at robot scale' as part of it, but leading to incorrect or even no answers to the observer when verifying interactions of some robots in some circumstances, whenever P is non-trivial.

If we relax the memory constraint for robots and allow them to have either 'limited' or 'unlimited' memory, then the proof of Theorem 2 remains valid, i.e. even if P-modules are just assumed to be 'realizable' (cf. Definition 7). This leads to the following result.

Corollary 2. *Let P be any non-trivial interaction property (for interacting robots). Suppose robots have no memory constraint. Then there is* no *P-module that always answers correctly, i.e. there is no P-module, algorithmic or otherwise, that can always correctly decide at runtime, for all robots X and Y and situations Z, whether the action taken by X towards Y when X (in any state) interacts with Y (in its sensed state) while Y is facing situation Z, satisfies property P or not.*

The result resembles that of Theorem 1, which was specifically proved for robot properties in this case. The difference is that Theorem 1 was proved for

properties of robot *programs*, and Corollary 2 for properties of robot *interactions* and that is what we need here. Note that the proof of Theorem 1 relied on the undecidability of the *Halting Problem*, whereas the proof of Theorem 2 and Corollary 2 did not. As a consequence, Corollary 2 enabled us to make a stronger claim than Theorem 1, giving us a general *impossibility* result for the case of interaction properties.

5.3 Some consequences

Theorem 2 has immediate implications for observers that want to make use of (feasible, algorithmic) modules for verifying properties of robot interaction. Considering the question from the beginning of Section 4:

> *Can an observer always tell from inspecting and monitoring their programs whether the interactions between any two robots will always satisfy a given property P (at runtime), such as a specified set of rules of law or ethics, or any other [non-trivial] set of formally expressed constraints?*

we can now conclude that, *for properties P that are non-trivial to the observer*, the answer is always '*no*': any feasible (deterministic) module for verifying P, even if it is not assumed to be algorithmic, will necessary fail on some robots, in some conceivable circumstances.

This also answers the quest for verification tools, algorithmic or otherwise, in the many cases of interest that we encountered. For example, we can easily conclude now that, in general, observers cannot rely on an *ethical governor*, i.e. an effective (algorithmic) means to verify that the interactions of given robots are always sure to be *ethical*.

Corollary 3. *There is no (feasible) deterministic module, algorithmic or otherwise, that always correctly decides, for all robots X and Y, whether the action taken when robot X interacts with robot Y in a given situation is* ethical *or not.*

Proof. It follows from Example 5 that $P \equiv \mathit{ethical}$ is a non-trivial interaction property (in general). Now apply Theorem 2. □

A similar corollary may be obtained for $P \equiv \mathit{legal}$, another property of robot interaction that is non-trivial to observers 'in general' (cf. Example 6).

Finally, we remark that Theorem 2 is not an 'if and *only* if' result, i.e. the 'non-triviality' of a property P for some class of circumstances is a sufficient, but not a necessary condition for the impossibility of P-testing modules. It is likely that (many) other classes of detectable circumstances can be characterised for which all presumed (feasible, possibly algorithmic) P-testing modules must eventually fail.

Note that the notion of 'non-triviality' as defined in Section 4.2, is intrinsically *observer-dependent*. Thus, Theorem 2 may also be interpreted as saying that an observer can *only* have an (algorithmic) module to verify a given property P 'at runtime' when P is *trivial to this observer in some circumstances*, i.e. when, for all situations Z on the H-list, there are at least some robots X and Y such that, when X targeting Y while Y is facing Z, the meaningful

'instructions' which X could possibly send to Y in this case either all satisfy P, or they do not satisfy it.

Example 7. Let $P \equiv$ *ethical*. Assume that, in some application, the observer has an (e.g. algorithmic) module that enables it to verify 'at runtime' whether the interactions between one robot and another one are ethical or not. Then Theorem 2 implies that, to this observer, *'ethicality must be trivial in some circumstances'* in this application, i.e. for all situations $Z \in H$, there must be some robots X and Y such that, when X is targeting Y and Y is facing situation Z, all *meaningful* 'instructions' X could possibly send to Y in this case are either all ethical or all non-ethical. Similar considerations apply when $P \equiv$ *legal*.

6 Verifying non-trivial interaction properties - discussion

There are various ways to look at the result we obtained. One might say, for example, that it suggests that 'testing whether (human) interactive behaviour will be ethical in all circumstances is inherently impossible', which one could have suspected to begin with! However, we are not talking about (human) ethics in general, but about *machine ethics*.

In machine ethics one considers the behaviour of *artefacts* like autonomous robots, as we do in this report. If their behaviour is to follow a code of ethics, we may assume that this code is somehow implemented in their governing control unit. With the increasing use of autonomous robots in daily life, it is important that it should be *verifiable* that their programs satisfy the properties that their makers promised, in all situations.

Our main result shows that, in general, this requirement of verifiability cannot be fulfilled, for any non-trivial property of robot interaction, if we want to achieve it in a way that can be automated. More precisely, no computer program (not even any oracle for that matter) exists that can do this kind of testing at runtime without ultimately running into cases where it will fail. The result is proved as a theoretical impossibility statement.

a) Modelling The question whether properties of interaction can always be verified (algorithmically or otherwise), needs various steps of analysis. In order to approach the question in sufficiently formal terms, it is necessary to define a 'programming framework' in which it is meaningful to consider the possibility or impossibility of certain algorithms.

To begin with, various assumptions must be made on how robots can 'sense' and process the situations around them, on how they can interact with each other, and on how this can be captured in terms of programs. In the programming framework we defined, we left many details 'implicit' and focussed mostly on one kind of interactions (*targeting*) and the corresponding instructions for it, for the sake of argument.

This led to a generic programming system for autonomous robots, of sufficient generality. With this framework in place, questions about the existence or non-existence of certain verifying modules make sense. The result is certainly dependent on the formalism but its generality is important.

b) Verification The next ingredient is a sufficiently formal view of 'properties' (of interaction) and of what one actually wants to verify. Given a desirable property P like *ethicality*, the required testability of our robot software suggests

that the following predicate M_P on 4-tuples $\langle X, Y, q_Y, S_Y \rangle$ should be considered for testing:

> $M_P\langle X, Y, q_Y, S_Y \rangle \equiv$ *does the interaction which occurs when a robot of sort* X *(in any state) is targeting a robot of sort* Y *(in state* q_Y *as sensed by* X*), which is facing situation* $Z = S_Y$ *with* Z *on the list of recognized 'exceptional' situations, satisfy property* P*?*

It is tacitly assumed that the predicate makes sense, i.e. that the parameters are all effectively given and 'known' at runtime (to robot X, to the observer, and to the P-module), and that P is defined in terms that are 'understood' as well (as for all codified properties). If an observer can test robots interactions for property P, then it is reasonable to assume that the observer should be able to answer queries like M_P for any given instance of $\langle X, Y, q_Y, S_Y \rangle$.

In this report we focused on the reasonable assumption that an observer would want to delegate the answering of these queries to a computer algorithm or even an *oracle*, a P-module, that is tuned to its interpretation of P in all circumstances. The crucial question then becomes: do P-modules always exist, for all properties P of interest? This, as we have shown, turns out *not* to be the case in our model.

c) Hard properties An interaction property P may be 'hard' to verify when any interacting robot, when confronted with a suitable circumstance, can act (potentially) in several meaningful ways, of which some satisfy P and some do not. Properties that allow for these kind of options were called 'non-trivial'.

In the case of 'non-trivial' properties, a P- module would have to be able to 're-construct' the 'inner argument' of a robot X when it decides its action towards a robot Y, taking any further information into account that it has, in order to decide P for the action X decides to take. If a P-module could do this, then nothing prevents a robot from utilizing the very module in its decisioning and possibly, decide *against* what the P-module had thought. We have shown that it is possible to 'program' this, implying that P-modules for all non-trivial properties must occasionally be *wrong* or *default*.

d) Result The result shows that, whenever P is non-trivial, *no* P-module, algorithmic or otherwise, exists that always answers M_P-queries correctly. In other words, as a predicate, M_P queries cannot be decided by any means whenever P is non-trivial. In this way it may be compared to Rice's Theorem in computability theory that proves undecidability for all non-trivial semantic properties of classical 'programs', now in the different programmatic context of robot interactions and with an even stronger conclusion.

The case of $P \equiv$ *ethical* is a special instance of the result. Simply stated it can be interpreted as saying that, in general, there is no module (algorithmic or otherwise) for deciding 'at runtime' whether interacting autonomous robots always satisfy their code of ethics while interacting, i.e. in all circumstances. It is as close as we get to proving that a 'general observer' $\mathcal{O}$ that could judge whether the behaviour of any robot A towards another robot is always ethical, under the assumptions as we made them, cannot exist.

Non-triviality thus appears as an important notion in the analysis. On the one hand, it is the crucial source of the impossibility result. On the other hand,

it shows what must be avoided when designing artefacts whose behavioural properties have to be verifiable at runtime. For example, in the case of ethical laws, it gives a compelling reason why such laws must not allow for the existence of *meaningful* alternative opinions. (This is an immediate consequence of the definition of non-triviality, cf. Definition 5.)

Example 8. An example is the solution of the *Trolley Problem* [40] in which preference should always be given to the passengers' survival. Other examples of well-designed rules are the rules for chess, checkers, sport games, etcetera.

e) Interaction In the conceptual elaboration of the problem, several ingredients were left intentionally 'unspecified', for the sake of generality. For example, we have left it open what (other) interactions can concretely occur when a robot is interacting with another robot, notably when it is facing some particularly relevant or challenging external situations. Theoretically, the result we prove is related to the well-known result for 'classical' programs that, in general, there is no algorithmic procedure for deciding which instructions of a program are actually executed and which are not.

For robot programs in our framework, it turns out that 'non-trivial' program properties are just as undecidable as in the case of classical programs. However, this does not quite cover properties of the 'interaction' between them, like *ethicality*. This warranted a general look at what may happen 'between' robots. Indeed, all that matters here is that it can somehow be tested whether any kind of interaction that takes place, satisfies the property P under consideration.

f) On assumptions The result we obtained depends on several assumptions and idealizations. For example, in the proof of Lemma 1 it was assumed that the robot programming framework is 'closed' under the simple form of program composition that was needed there. If this is not allowed, then we would not have a sufficiently universal programming framework and systematic verification of program properties would be difficult in any case.

We also assumed that the programming framework has constructs that let robots deal with the input from their sensory and motor units and, conversely to pass instructions to them. The precise way in which this data is passed in structured variables ('situations') and how a robot acts with it, does not really matter. The way we allowed a robot to 'sense' the data from a robot it is observing, is designed for the purpose of exposition but is not unrealistic. It is well-supported by the recent conceptions of cyber-physical systems as being *minimally machine conscious* [37].

Finally, we note that our result does *not* rely on unrealistic assumptions about the robot hardware. In particular, in the crucial argumentation we merely needed to assume that robots have a bounded working memory available, i.e. bounded in terms of the size of their program and situational input data.

7 Conclusion

Advanced industrial robots, e-vehicles of all kinds, and other autonomous robots are increasingly being used in our working and living environments. It has led to an increased interest in the verifiability of the programs that control

these robots and in ways of preventing the robots from causing harm. As a protection against this, it has been widely suggested that autonomous robots should be programmed to obey suitable ethical and legal constraints (cf. [24]). This consideration has led us to the following key question:

> *Can an observer always tell from inspecting and monitoring a robot's program whether the robot will always obey the given rules of law or ethics, or any other set of formally expressed constraints, in any interaction with other robots (or humans)?*

In this report, we have sketched an idealized model of programmed robot interaction that allowed us to formalize the problem. In particular, it enabled us to formulate decision problems about the testability of properties like *ethicality* in ensembles of robots that interact. We specialized the problem to the testability of predicates M_P, with M_P being satisfied for a 4-tuple $\langle X, Y, q_Y, S_Y \rangle$ if and only if the interaction that is triggered when a robot of type X (in any state) is targeting a robot of type Y (in the state q_Y sensed by X) that is facing a recognized exceptional situation S_Y, satisfies P.

We proved that, for all non-trivial interaction properties P, there can be *no* feasible modules, algorithmic or otherwise, that can always 'decide' predicate M_P, i.e. for all robots X and Y and in all circumstances. The result gives theoretical evidence that the formal verification of these properties will be hard, if not impossible, in general. It also confirms, at least partially, the claim of Charisi *et al.* [8] that *'full formal verification is likely to be unrealistic [. . .] both because of non-symbolic components and because of practical complexity'*.

The result applies in particular to the verification of ethical and legal decisioning. It means that, if ethical or legal properties of robot software are to be verified in practice, one should either eliminate the need of knowing non-trivial properties or limit the extent of the programs that are allowed, as for ordinary programs. It is conceivable that many other questions about the general verification of properties for robots can be cast in the same framework.

References

1. M. Anderson, S.L. Anderson (eds.) *Machine Ethics*, Cambridge University Press, 2011
2. I. Asimov, Runaround, *Astounding Science Fiction*, 29:1 (1942) 94-103 (reprinted in several later collections of Asimov's robot stories)
3. E. Awad, S. Dsouza, R. Kim, J. Schulz, J. Henrich, A. Shariff, J-F. Bonnefon, I. Rahwan, The Moral Machine experiment, *Nature* 563 (2018) 59-64
4. E. Bird, J. Fox-Skelly, N. Jenner, R. Larbey, E. Weitkamp, A. Winfield, *The ethics of artificial intelligence: Issues and initiatives*, Study PE 634.452, Scientific Foresight Unit, Panel for the Future of Science and Technology, DG Parliamentary Research Services, European Parliament, March 2020
5. N. Bostrom, E. Yudkowsky, The Ethics of Artificial Intelligence, in: K. Frankish, W.M. Ramsey (Eds), *Cambridge Handbook of Artificial Intelligence*, Cambridge University Press, New York, 2014, Ch 14, pp. 316-334
6. P. Bremner, L.A. Dennis, M. Fisher, A.F. Winfield, On proactive, transparent and verifiable ethical reasoning for robots, *Proc. IEEE* 107:3 (2019) 541-561
7. J-A. Cervantes, L-P. Rodriguez, S. Cervantes, F. Cervantes, F. Ramos, Artificial Moral Agents: A Survey of the Current Status, *Science and Engineering Ethics* 26 (202) 501-532

8. V. Charisi,L. Dennis, M. Fisher, R. Lieck, A. Matthias, M. Slavkovic, J. Sombetzki, A.F.T. Winfield, R. Yampolski, Towards Moral Autonomous Systems, *arXiv*:1703.04741[cs.AI], 2017
9. COMEST, *Report on robotics ethics*, Working Group Report SHS/YES/COMEST-10/17/2 REV., Commission mondiale d'éthique des connaissances scientifiques et des technologies (COMEST), UNESCO, September 2017, `https://unesdoc.unesco.org/ark:/48223/pf0000253952`
10. L. Dennis, M. Fisher, M. Slavkovic, M. Webster, Formal verification of ethical choices in autonomous systems, *Robotics and Autonomous Systems* 77 (March 2016) 1-14
11. P.J. Denning, D.E. Denning, Dilemmas of Artificial Intelligence, *Comm. ACM* 63:03 (2020) 22-24
12. V. Dignum *et al.*, Ethics by Design: Necessity or Curse?, in: *Proc. 2018 AAAI/ACM Conference on AI, Ethics, and Society* (AIES'18), 2018, pp. 60-66
13. E.W. Dijkstra, Guarded commands, non-determinacy and formal derivation of programs, *CACM* 18:8 (1975) 453-457
14. M. Englert, S. Siebert, M. Ziegler, Logical Limitations to Machine Ethics with Consequences to Lethal Autonomous Weapons, *arXiv*:1411.2842 [cs.CY], 11 Nov. 2014
15. N.J. Goodall, Ethical Decision Making during Automated Vehicle Crashes, *Transportation Research Record: Journal of the Transportation Research Board* 2424:1 (2014) 58-65
16. J. Guiochet, M. Machin, H. Waeselynck, Safety-critical advances robots: A survey, *Robotics and Autonomous Systems* 94 (August 2017) 43-52
17. T. Holstein, G. Dodig-Crnkovic, P. Pellicione, Ethical and Social Aspects of Self-Driving Cars, *arXiv* 1802.04103, 2018
18. M. James, Halting Problem Used To Prove A Robot Cannot Computably Kill A Human, *i-Programmer News*, 26 Nov. 2014, `https://www.i-programmer.info/news/112-theory/8000-halting-problem-used-to-prove-a-robot-cannot-computably-kill-a-human.html`
19. R. Krzanowski, K. Mamak, K. Trombik, E. Gradzka, Is Machine Ethics computable, non-computable, or nonsensical?, in: *Machine Ethics and Machine Law*, E-Proceedings, Copernicus Center, Jagiellonian University, Cracow, November 2016, `https://maschinenethik.net/wp-content/uploads/2016/11/PROCEEDINGS_MEML_2016.pdf`
20. G.A. Lanzarone, F. Gobbo, Is Computer Ethics Computable?, in: T.W. Bynum, M. Calzarossa, I. De Lotto, S. Rogerson (Eds.), *Living, Working and Learning Beyond Technology*, Conference Proc. ETHICOMP 2008, University of Pavia, Mantua (It.), 2008, pp. 530-539
21. R. Leenes, E. Palmerini, B-J. Koops, A. Bertolini, P. Salvini, F. Lucivero, Regulatory challenges of robotics: some guidelines for addressing legal and ethical issues, *Law, Innovation and Technology* 9:1 (2017) 1-44
22. S. Leigh Anderson, Asimov's "three laws of robotics" and machine metaethics,*AI & Society* 22 (2008) 477-493
23. J. Leikas, R. Koivisto, N. Gotcheva, Ethical Framework for Designing Autonomous Intelligent Systems, *J. Open Innovation: Technology, Markets, and Complexity* 5:1 (2019) article 18 (12 pages)
24. P. Lin, K. Abney, G. Bekey, Robot ethics: Mapping the issues for a mechanized world, *Artificial Intelligence* 175: 5-6 (2011) 942-949
25. J. Loh, Responsibility and Robot Ethics: A Critical Overview, *Philosophies* 2019, 4:4, 58, `https://doi.org/10.3390/philosophies4040058`
26. M. Luckcuck, M. Farrell, L.A. Dennis, C. Dixon, M. Fisher, Formal Specification and Verification of Autonomous Robotic Systems: A Survey, *ACM Computing Surveys* 52:5 (2019) 1-41
27. B. Mermet, G. Simon, Formal Verification of Ethical Properties in Multiagent Systems, in: G. Bonnet, M. Harbers, K.V. Hindriks, M. Katell, C. Tessier (Eds.), *Ethics in the Design of Intelligent Agents*, Proc. 1st Workshop (EDIA 2016), part of ECAI 2016, CEUR Workshop Proceedings Vol-1668, 2016, pp. 26-31
28. J.H. Moor, Is Ethics Computable, *Metaphilosophy* Vol. 26:1/2 (1995) 1-21
29. J.H. Moor, The nature, importance, and difficulty of machine ethics, *IEEE Intelligent Systems* 21 (2006) 18-21

30. V. Murashov, F. Hearl. J. Howard, Working Safely with Robot Workers: Recommendations for the New Workplace, *J. Occup. Environ. Hyg.* 13:3 (2016) D61-D71.
31. N. Nevejans, European Civil Law Rules in Robotics, Study PE 571.379, Policy Department C: Citizen's Rights and Constitutional Affairs, DG Internal Policies, European Parliament, October 2016
32. T.M. Powers, Prospects for a Kantian Machine, *IEEE Intelligent Systems* 21:4 (2006) 46-51
33. H. Rogers Jr., *Theory of recursive functions and effective computability*, McGraw-Hill, New York, 1967
34. A. Sharkey, Can we program or train robots to be good, *Ethics and Information Technology*, 2017, `https://doi.org/10.1007/s10676-017-9425-5`
35. V. Vakkuri, K-K. Kemell, J. Kultanen, P. Abrahamsson, The Current State of Industrial Practice in Artificial Intelligence Ethics, *IEEE Software* 37:4 (2020) 50-57
36. W. Wallach, C. Allen, *Moral Machines: Teaching Robots Right from Wrong*, Oxford University Press, 2009
37. J. Wiedermann, J. van Leeuwen, Towards Minimally Conscious Finite-State Controlled Cyber-Physical Systems: A Manifesto. In: T. Bureš *et al.* (Eds.), *SOFSEM 2021: Theory and Practice of Computer Science*, Proc. 47th Int. Conference on Current Trends in Theory and Practice of Computer Science, Lecture Notes in Computer Science, Vol 12607, Springer, 2021, pp. 43-55
38. J. Wiedermann, J. van Leeuwen, Towards minimally conscious cyber-physical systems - a design philosophy, *Technical report* UU-PCS-2020-02, Center for Philosophy of Computer Science, Dept. of Information and Computing Science, Utrecht University, Utrecht, 2020 (extended version of [37])
39. Wikipedia, Three Laws of Robotics, `https://en.wikipedia.org/wiki/Three_Laws_of_Robotics`
40. Wikipedia. Trolley problem, `https://en.wikipedia.org/wiki/Trolley_problem`
41. A.F. Winfield, K. Michael, J. Pitt, V. Evers, Machine ethics: The design and governance of ethical AI and autonomous systems, *Proc. IEEE* 107:3 (2019) 509-517
42. R.V. Yampolskiy, Unpredictability of AI: On the Impossibility of Accurately Predicting All Actions of a Smarter Agent, *Journal of Artificial Intelligence and Consciousness* 07:01 (2020) 109-118