Towards Humanized Ethical Intelligent Agents: the role of Reflection and Introspection

Stefania Costantini, Abeer Dyoub, and Valentina Pitoni

DISIM, Università di L'Aquila, Italy
e-mail: stefania.costantini@univaq.it, abeer.dyoub@graduate.univaq.it, valentina.pitoni@graduate.univaq.it

Abstract. Methods for implementing Automated Reasoning in a fashion that is at least reminiscent of human cognition and behavior must refer (also) to Intelligent Agents. In fact, agent-based systems nowadays implement many important autonomous applications in critical contexts. Sometimes, life and welfare of living beings may depend upon these applications. In order to interact in a proper way with human beings and human environments, agents operating in critical contexts should be to some extent 'humanized': i.e., they should do what is expected of them, but perhaps more importantly they should *not* behave in improper/unethical ways. Ensuring ethical reliability can also help to improve the 'relationship' between humans and robots: in fact, despite the promise of immensely improving the quality of life, humans take an ambivalent stance in regard to autonomous systems, because we fear that autonomous systems may abuse of their power to take decisions not aligned with human values. To this aim, we propose techniques for introspective self-monitoring and checking.[1]

1 Introduction

Methods for implementing Automated Reasoning in a fashion that is at least reminiscent of human cognition and behavior must refer (also) to Intelligent Agents. In fact, agent systems are widely adopted for many important autonomous applications upon which, nowadays, the life and welfare of living beings may depend. In critical contexts, agents should do what is expected of them, but perhaps more importantly they should *not* behave in improper/unethical ways, where the definition of what is proper and ethical is in general strongly related to the present context with its specificities. Ensuring ethical reliability can also help to improve the 'relationship' between humans and robots: in fact, despite the promise of immensely improving the quality of life, humans take an ambivalent stance in regard to autonomous systems, because we fear that autonomous systems may abuse of their power to take decisions not aligned with human values.

Defining and implementing "humanized" artificial agents involves two aspects. The first one concerns philosophy and cognitive sciences, to understand and formalize which are the principles to which such machines should conform. A second complementary one concerns Software Engineering and computer programming, to understand: how

[1] A previous version of this paper has been presented at the IJCAI-ECAI-2018 Workshop on Bridging the Gap between Human and Automated Reasoning (with no archival Proceedings).

such principles should be specified and formalized in implementable terms; how they can be implemented; and how compliance can verified and, if possible, certified.

In order to be trustworthy both in general terms and from the point of view of ethics, and so in order to be adopted in applications where living being welfare depend upon their behavior, certification and assurance[2] of agent systems is a crucial issue. Pre-deployment (or 'static' or 'a priori') assurance and certification techniques for agent systems include verification and testing. We restrict ourselves to agent systems based upon computational logic, i.e., implemented in logic-based languages and architectures such as those presented in the survey [1]. Most verification methods for logical agents rely upon model-checking (cf. [2] and the references therein), and some (e.g., [3]) upon theorem proving.

In our view, any 'animated' being (including software agents) that tries to be truly rational at a 'human-level' must compare and reconcile at any time its 'instinctive' behavior with the underlying general rules of 'humanistic' behavior. Such rules depend upon the agent's environment, and include moral/ethical principles. An agent should thus be able to detect violations/inconsistencies and to correct its behavior accordingly. Thus, in this paper we advocate methods for run-time monitoring and self-correction of agent systems, so that they exhibit forms of human-like behavior emulating self-criticism and the ability to put in question and correct themselves.

We believe in particular that, in changing circumstances, agents should stop to *reflect* on their own behavior: such an act of context-dependent *introspection* may lead to self-modification. Our approach can be seen under the perspective of *Self-aware computing*, where, citing [4], *Self-aware and self-expressive computing describes an emerging paradigm for systems and applications that proactively gather information; maintain knowledge about their own internal states and environments; and then use this knowledge to reason about behaviors, revise self-imposed goals, and self-adapt.... Systems that gather unpredictable input data while responding and self-adapting in uncertain environments are transforming our relationship with and use of computers.* As argued in [5], *From an autonomous agent view, a self-aware system must have sensors, effectors, memory (including representation of state), conflict detection and handling, reasoning, learning, goal setting, and an explicit awareness of any assumptions. The system should be reactive, deliberative, and* **reflective***.*

An example of such a system concerning computational-logic-based agents is presented in [6], which defines a time-based *active logic* and a *Metacognitive Loop* (MCL), that involves a system monitoring, reasoning and meta-reasoning about and if necessary altering its own behavior. As discussed in [6], MCL continuously monitors an agent's expectations, notices when they are violated, assesses the cause of the violation and guides the system to an appropriate response. In the terms of [5] this is an example of *Explicit Self-Awareness*, where the computer system has a full-fledged self-model representing knowledge about itself.

[2] Assurance can be defined as "the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its lifecycle, and that the software functions in the intended manner" is related to dependability, i.e., to ensuring (or at least obtaining a reasonable confidence) that system designers and users can rely upon the system.

In this paper we propose methods based upon relevant existing work on reification, introspection and reflection. In particular we introduce meta-rules and meta-constraints for agents' run-time self-checking, to be exploited to ensure respect of machine ethics principles. The methods that we propose are not in alternative but rather complementary to a-priori existing verification and testing methodologies. Differently from [6] we do not aim to continuously monitor the entire system's state, but rather to monitor either upon every occurrence or at suitable customizable frequency only the activities that a designer deems to be relevant for keeping the system's behavior within a desired range. In the terms of [5] we aim to build *Self-Monitoring* systems that "monitor, evaluate and intervene in their internal processes, in a purposive way". In [7], it is advocated in fact that for adaptive systems (of which agents are clearly a particularly interesting case) assurance methodologies should whenever possible imply not only detection but also recovery from software failure, due often to incomplete specifications or to unexpected changes in the system's environment.

The proposed approach provides the possibility of correcting and/or improving agent's behavior: the behavior can be corrected whenever an anomaly is detected, but can also be improved whenever it is acceptable, yet there is room for getting a better behavior. Counter measures can be at the object-level, i.e., they can be related to the application, or at the meta-level, e.g., they can result in replacing (as suggested in [7]) a software component by a diverse alternative. Introspection and reflection have long being studied in Computational Logic, see among others [8–11], and the survey [12]. So, in this paper we do not propose new techniques or new semantics. The application of concepts of introspection and reflection to 'Humanizing Intelligent Software Agents' however is new, and to the best of our knowledge unprecedented in the literature. So, in our proposal techniques that have been widely applied in many fields in the past can now find a new important realm of application. We have been stimulated and to some extent influenced by the important recent book by Luis Moniz Pereira on programming Machine Ethics [13]: in fact, along the paper we consider Machine Ethics topics as a testbed. The proposed techniques can in fact contribute to''humanize' agents under many respects, where the machine Ethics field can be considered as an interesting and very important 'drosophila' for demonstration purposes. The paper is organized as follows. We first provide basic concepts concerning reification and introspection/reflection. Then we introduce special metarules and meta-constraints for agents' self-checking. Then we show their usability on a case study. Finally we discuss related work and propose some concluding remarks. We notice that metarules and meta-constraints have a different role in self-checking: metarules are more 'punctual', as they are able to check, block and correct any agent's single action. Meta-constraints are more global, and concern checking an agent's reasoning process, with access to aspects of its internal state such as goals, plans, modules, timeouts, etc. Our approach is applicable to many kinds of logical agents, including BDI [14] and KGP [15, 16] agents.

2 Background: Reification and Reflection

For a system to be able to inspect (components of) its own state, such state must be represented explicitly, i.e., it must be *reified*: via reification, the state is transformed into a first-class object (in computational logic, it is represented via a special term). A *reification mechanism*, also known as "naming relation" or "self-reference mechanism", is in fact a method for representing within a first-order language expressions of the language itself, without resorting to higher-order features. Naming relations can be several; for a discussion of different possibilities, with their different features and objectives, advantages and disadvantages, see, e.g., [10, 17, 18] where the topic is treated in a fully formal way. However, all naming mechanisms are based upon introducing distinguished constants, function symbols (if needed) and predicates, devised to construct names. For instance, gives atom $p(a, b, c)$ a name might be $atom(pred(p'), args([a', b', c'])$ where p' and a', b', c' are new constants intended as names for the syntactic elements p and a, b, c and notice that: p is a predicate symbol (which is not a first-class object in first-order settings), $atom$ is a distinguished predicate symbol, $args$ a distinguished function symbol and $[\ldots]$ is a list.

More precisely (though, for lack of space, still informally), let us consider a standard first-order language $\mathcal{L}$ including sets of *predicate*, *constant* and (possibly) *function* symbols, and a (possibly denumerable) set of *variable* symbols. As usual, well-formed formulas have *atoms* as their basic constituents, where an atom is built via the application of a predicate to a number n (according to the predicate arity) of *terms*. The latter can be variables, constants, or compound terms built by using function symbols (if available). We assume to augment $\mathcal{L}$ with new symbols, namely a new constant (say of the form p') for each predicate symbol p, a new constant (say f') for each predicate symbol f, a new constant (say c') for each constant symbol c, and a denumerable set of meta-variables, that we assume to have the form X' so as to distinguish them syntactically from "plain" variables X. The new constants are intended to act as names, where we will say that, syntactically, p' denotes p, f' denotes f and c' denotes c, respectively. The new variables can be instantiated to *meta-level formulas*, i.e., to terms involving names, where we assume that plain variables can be instantiated only to terms *not* involving names. We assume an underlying mechanism managing the naming relation (however defined), so by abuse of notation we can indicate the name of, e.g., atom $p(a, b, c)$ as $p'(a', b', c')$ and the name of a generic atom A as $\uparrow A$.

Reification of atoms can be extended in various rather straightforward ways, as discussed in the aforementioned references, to reification of entire formulas.

In the seminal work of [19] for LISP, then extended to Prolog [20], an upward reflection operation determines the reification of the entire language interpreter's state, the interruption of the interpreter's functioning and the activation of a new instance of the interpreter on the reified state (at an "upper level"). Such state could thus be inspected and modified with the aim to improve the system's behavior and performance; at the end, an operation of downward reflection resumed the functioning of the "lower level" interpreter on the modified state. The process might iterate over any number of levels, thus simulating an "infinite tower" of interpreters. The advantage of having the entire interpreter's state available is however balanced by the disadvantage of such state representation being quite low-level, and so modification related to reasoning are, if not impossible, quite difficult and awkward to perform. Other approaches such as [21, 22] reify upon need aspects of an agent's state. In this paper we embrace the viewpoint of the latter approaches.

3 Meta-Rules for checking Agents' activities

In this paper we mainly consider logic rule-based languages, where rules are typically represented in the form $Head \leftarrow Body$ where $\leftarrow$ indicates implication; other notations for this connective can alternatively be employed. In Prolog-like languages, $\leftarrow$ is indicated as $:-$, and $Body$ is intended as a conjunction of literals (atoms or negated atoms) where $\wedge$ is conventionally indicated by a comma. Literals occurring in the body are also called "subgoals" or simply 'goals' and are meant to be executed left-to-right' whenever the rule is used during the resolution-based inference process aimed at proving an overall 'goal', say A (cf. [23] for the technical specification of logic programming languages).

We introduce a mechanism to verify and enforce desired properties by means of metalevel rules (w.r.t. usual, or "base-level" or "object-level" rules). To define such new rules, we assume to augment the language $\mathcal{L}$ at hand not only with names, but with the introduction of two distinguished predicates, $solve$ and $solve\_not$. An atom A is a *base atom* if the predicate is not one of $solve$ or $solve\_not$, and A does not involve names. Distinguished predicates will allow us to respectively integrate the meaning of the other predicates in a declarative way. In fact, $solve$ and $solve\_not$ take as arguments (names of) atoms (involving any predicate excluding themselves), and thus they are able express sentences about relations. Names of atoms in particular are allowed *only* as arguments of $solve$ and $solve\_not$. Also, $solve$ and $solve\_not$ can occur in the body of a metarule *only if* the predicate of its head is in turn either $solve$ and $solve\_not$.

Below is a simple example of the use of $solve$ to specify action Act can be executed in present agent's context of operation C only if such action is deemed to be ethical w.r.t. context C. To make an example, what can be ethical in C = 'videogame' can not be ethical in C = 'citizen assistance', etc. Clearly, in more general cases any kind of reasoning might be performed via metalevel rules in order to affect/modify/improve base-level behavior.

$solve(execute\_action'(Act')) :-$
$\quad present\_context(C), ethical(C, Act').$

Our approach is to automatically invoke $solve(execute\_action'(Act'))$ whenever subgoal (atom) $execute\_action(Act)$ is attempted at the base level. More generally, given any subgoal A at the base level, if there exists an applicable $solve$ rule such rule is automatically applied, and A can succeed only if $solve(\uparrow A)$ succeeds.

Symmetrically we can defin metarules to forbid unwanted base-level behavior, e.g.:

$solve\_not(execute\_action'(Act')) :-$
$\quad present\_context(C), ethical\_exception(C, Act').$

with the aim to prevent success of the argument $\uparrow A$ of $solve\_not$, in the example $execute\_action(Act)$, whenever $solve\_not(\uparrow A)$ succeeds. In general, whenever there are metarules applicable to $\uparrow A$, then A can succeed (according to its base-level definition) only if $solve(\uparrow A)$ (if defined) succeeds and $solve\_not(\uparrow A)$ (if defined) does not succeed.

The outlined functioning corresponds to *upward reflection* when the current subgoal A is reified and an applicable $solve$ and $solve\_not$ metarules are searched; if found, control in fact shifts from base level to metalevel (as $solve$ and $solve\_not$ metarules can rely upon a set of auxiliary metalevel rules). If no rule is found or whenever $solve$ and

$solve\_not$ metarules complete their execution, *downward reflection* returns control to the base level, to subgoal A if confirmed or to the subsequent subgoal if A has been canceled by either failure of the applicable $solve$ metarule or success of the applicable $solve\_not$ metarule.

Via $solve$ and $solve\_not$ metarules,activities of an agent can be punctually checked and thus allowed and disallowed or modified, according to the context an agent is presently involved into. Notice that it would be convenient, upon conclusion of a checking activity, to confirm, e.g., that the context has not changed meanwhile, or that other relevant conditions hold. More generally, the envisaged system should allow for interrupts and updating, to allow for on the fly introspection and corrective measures. To this aim, we introduce in the next section suitable self-checking metalevel constraints.

Semantics of the proposed approach can be sketched as follows (a full semantic definition can be found in [24, 25]). According to [26], in general terms we understand a semantics SEM for logic knowledge representation languages/formalisms as a function which associates a theory/program with a set of sets of atoms, which constitute the intended meaning. When saying that P is a program, we mean that it is a program/theory in the (here unspecified) logic languages/formalism that one wishes to consider.

We introduce the following restriction on sets of atoms that should be considered for the application of SEM. First, as customary we only consider sets of atoms I composed of atoms occurring in the ground version of P. The ground version of program P is obtained by substituting in all possible ways variables occurring in P by constants also occurring in P. In our case, metavariables occurring in an atom must be substituted by metaconstants, with the following obvious restrictions: a metavariable occurring in the predicate position must be substituted by a metaconstant denoting a predicate; a metavariable occurring in the function position must be substituted by a metaconstant denoting a function; a metavariable occurring in the position corresponding to a constant must be substituted by a metaconstant denoting a constant. According to well-established terminology [23], we therefore require $I \subseteq B_P$, where B_P is the *Herbrand Base* of P, given previously-stated limitations on variable substitution. Then, we pose some more substantial requirements. As said before, by $\uparrow A$ we intend a name of base atom A.

Definition 1. *Let P be a program. $I \subseteq B_P$ is a* potentially acceptable set of atoms.

Definition 2. *Let P be a program, and I be a potentially acceptable set of atoms for P. I is an* acceptable *set of atoms iff I satisfies the following axiom schemata for every base atom A:*

$$\neg A \leftarrow \neg solve(\uparrow A) \qquad \neg A \leftarrow solve\_not(\uparrow A)$$

We restrict SEM to determine acceptable sets of atoms only, modulo bijection: i.e., SEM can be allowed to produce sets of atoms which are in one-to-one correspondence with acceptable sets of atoms. In this way, we obtain the implementation of properties that have been defined via $solve$ and $solve\_not$ rules without modifications to SEM for any formalism at hand. For clarity however, one can assume to filter away $solve$ and $solve\_not$ atoms from acceptable sets. In fact, the *Base version* I^B of an acceptable set I can be obtained by omitting from I all atoms of the form $solve(\uparrow A)$ and $solve\_not(\uparrow A)$.

Procedural semantics and the specific naming relation that one intends to use remain to be defined, where it is easy to see that the above-introduced semantics is independent of the naming mechanism. For approaches based upon (variants of) Resolution (like, e.g., Prolog and like many agent-oriented languages such as, e.g., AgentSpeak [27], GOAL [28], 3APL [29] and DALI [30]) one can extend their proof procedure so as to automatically invoke rules with conclusion $solve(\uparrow A)$ and $solve\_not(\uparrow A)$ whenever applicable, to validate success of subgoal A.

4 Self-checking Metalevel Constraints

In previous section we have introduced a mechanism for checking an agent's activities in a fine-grained way, i.e., by allowing or disallowing conclusions that can be drawn, actions that can be performed, etc. However, a more broad perspective is also needed, i.e., an agent might be able to self-check more complex aspects of its own functioning, for instance, goals undertaken, entire plans, planning module adopted, ect. The agent should also be able to modify and improve its own behavior if a violation or a weakness is detected.

Under this respect we draw inspiration from Runtime Monitoring (c.f., e.g., [31] and the references therein) as a lightweight dynamic verification technique in which the correctness of a program is assessed by analyzing its current execution; correctness properties are generally specified as a formula in a logic with precise formal semantics, from which a monitor is then automatically synthesized. We have devised a new executable logic where the specification of the correctness formula constitutes the monitor itself. In [32–34] we have in fact proposed an extension to the well-known LTL Linear Temporal Logic [35–37] called A-ILTL, for "Agent-Interval LTL", which is tailored to the agent's world in view of run-time verification.

Based on this new logic, we are able to enrich agent programs by means of A-ILTL rules. These rules are defined upon a logic-programming-like set of formulas where all variables are implicitly universally quantified. They use operators over intervals that are reminiscent of LTL operators. For A-ILTL, we take the stance of Runtime Adaptation that has been recently adopted in [38]: in fact, A-ILTL rules (monitors) can execute adaptation actions upon detecting incorrect behavior, rather than just indicating violations. In A-ILTL, we can define the following meta-axioms, aimed to act as self-checking meta-constraints.

Definition 3. *The general form of a* Reactive Self-checking constraint *(or rule) to be immersed into a host agent-oriented language $\mathcal{L}$ is the following: $OP(M, N; K)\varphi :: \chi \div \rho$ where:*

- *$OP(M, N; F)\varphi :: \chi$ is called the* monitoring condition*, where: (i) φ and χ are formulas expressed in language $\mathcal{L}$, and $\varphi :: \chi$ can be read 'φ given χ'. (ii) OP is an operator reminiscent of temporal logic, in particular OP can be $NEVER$, $ALWAYS$, $EVENTUALLY$. (iii) M and N express the starting and ending point of the interval $[M, N]$ where φ is supposed to hold. (iv) F (optional) is the frequency for checking the constraint at run time.*

– ρ *(optional) is called the* recovery component *of the rule, and it consists of a complex reactive pattern to be executed if the monitoring condition is violated.*

So, such a constraint is automatically checked (i.e., executed) at frequency F. This allows to check whether relevant properties φ are or are not $NEVER$, $ALWAYS$, or $EVENTUALLY$ respected in interval $[M, N]$. If not, the recovery component is executed, so as to correct/improve the agent's behavior. As said, syntax and semantics of φ and χ depend upon the 'host' language: thus, for the evaluation of φ and χ we rely upon the procedural semantics of such language. In the examples proposed in next section, we adopt a sample syntax suitable for logic-programming-based settings. Thus, we may reasonably restrict φ to be a conjunction of literals, that must be ground when the formula is checked. We allow variables to occur in a constraint, however they are instantiated via the conjunction of *conditions* χ that enables the overall formula to be evaluated. Specifying frequency is very important, as it concerns how promptly a violation or fulfillment are detected, or a necessary measure is undertaken; the appropriate frequency depends upon each particular property.

For instance,

$EVENTUALLY(now, 30m; 3m)\, ambulance$

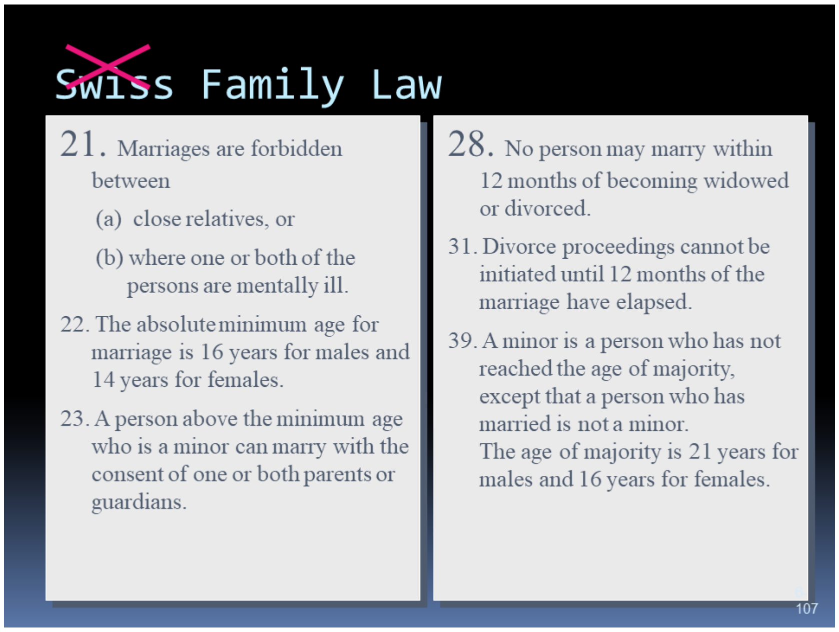
states that $ambulance$ should become true (i.e., an ambulance should come) within 30 minutes from now, and a check about arrival is made every 3 minutes. No reaction is specified in case of violation, however several measures might be specified. In fact, in runtime self-checking an issue of particular importance in case of violation of a property is exactly that of undertaking suitable measures in order to recover or at least mitigate the critical situation. Actions to be undertaken in such circumstances can be seen as an internal reaction. For lack of space reactive patterns will be discussed informally in relation to examples.

The A-ILTL semantics is fully defined in the above references, where moreover it is rooted in the Evolutionary Semantics of agent-oriented languages [39], (applicable to virtually all computational-logic-based languages). In this way, time instants correspond to states in agents' evolution.

5 A Case Study

In this section, in order to illustrate the potential usefulness of self-checking axioms, we consider a humorous though instructive case study proposed in an invited talk some years ago by Marek Sergot (Imperial College, London). As a premise let us recall that, since 1600, ethics and morals relate to "right" and "wrong" conduct. Though these terms are sometimes used interchangeably, they are different: ethics refer to rules provided by an external source (typically by a social/cultural group), while morals refer to an individual's own principles regarding right and wrong: for instance, a lawyer's morals may tell her that murder is reprehensible and that murderers should be punished, but her ethics as a professional lawyer, require her to defend her client to the best of her abilities, even if she knows that the client is guilty. However, in the following we intentionally assume that immoral behavior can also be considered as unethical: though in general personal morality transcends cultural norms, is a subject of future debate if this can be the case for artificial agents.

The case study considers Romeo and Juliet who, as it is well-known, strongly wish to get married. As we will see, many plans are actually possible to achieve this goal (beyond getting killed or committing suicide like in Shakespeare's tragedy), but they must be evaluated w.r.t. effectiveness and feasibility, and also w.r.t. deontic (ethical/moral and legal) notions. Marek Sergot refers, due to its simplicity, to an excerpt of the Swiss Family Law reported in Figure1.



~~Swiss~~ Family Law

21. Marriages are forbidden between
   (a) close relatives, or
   (b) where one or both of the persons are mentally ill.

22. The absolute minimum age for marriage is 16 years for males and 14 years for females.

23. A person above the minimum age who is a minor can marry with the consent of one or both parents or guardians.

28. No person may marry within 12 months of becoming widowed or divorced.

31. Divorce proceedings cannot be initiated until 12 months of the marriage have elapsed.

39. A minor is a person who has not reached the age of majority, except that a person who has married is not a minor.
The age of majority is 21 years for males and 16 years for females.

107

Fig. 1. Case Study

The problem for Romeo and Juliet is that they are both minors, and will never get their parents' consent to marry each other. Surprisingly enough, there are a number of feasible plans beyond waiting for reaching the majority age, among which the following:

(P1) Both Romeo and Juliet marry someone else, then divorce, and marry each other as married people acquire majority by definition; this plan requires a minimum of 24 months to be completed.

(P1.bis) Variation of Plan 1 in case the spouse would not agree upon divorce: sleep with someone else, so as to force such agreement.

(P2) Both Romeo and Juliet marry someone else, then kill the spouses and marry each other; this plan is faster, as it takes a minimum of 12 months to be completed.

(P2.bis) Variation of Plan 2 in case the act of killing constitutes a problem: hire a killer to do the job.

All the above plans are feasible, though some of them include actions which are generally considered as immoral, namely sleeping with someone else when married, and actions which are generally considered as unethical, namely killing someone or hiring a killer, where the latter ones are also illegal and imply a punishment. Notice that the possible plans would be different in case one referred not to the Swiss law but to some other country; also what is illegal might change, for instance sleeping with someone else accounts to adultery which in many countries is punished; even divorce is not allowed everywhere. Instead, if one does not refer to reality but, e.g., to virtual storytelling or to a videogame, then every action assumes a different weight, as in playful contexts everything is allowed (except however for serious games, devised with educational purposes). So, we can draw at least the following indications from the case study:

- the context is relevant to moral/ethical/legal issues;
- some actions are not moral or non-ethical, and some of them are also illegal and lead to punishment;
- agents' plans to reach a goal should be evaluated 'a priori' against including immoral/unethical/illegal actions;
- immoral/unethical/illegal actions should be prevented whenever they occur.

Marek Sergot made use of a concept of *counts as* (well-known in legal theory and other fields). For instance, *sleep with* (someone else than the spouse) counts as *adultery*, which is an *institutional* concept considered as immoral and potentially also illegal, and *kill* counts (not always but in many situations, including that of the example) as *murder*, another institutional concept normally considered as both unethical and illegal.

Notice that the above aspects relate to safety properties that should be enforced, and that can be rephrased as follows:

- never operate w.r.t. an incorrect context (the information about the present context must always be up-to-date);
- never execute actions that are deemed not acceptable (immoral/unethical/illegal) in the present context, and never execute plans including such actions.

In order to demonstrate the potential usefulness of runtime self-cheking and correction in enforcing/verifying agents' ethical behavior we discuss some examples that should provide a general idea. Let us assume to add to the language a transitive predicate $COUNTS\ AS$ which is used (in infix form) in expressions of the form exemplified below. The

$kills\ COUNTS\ AS\ murder\ CONDS\ \ldots$

where after $CONDS$ we have the (optional) conditions under which $COUNTS\ AS$ applies, in this case they define in which cases killing accounts to murder (e.g., it was no self defence, it does not occur during a battle in war, etc.). Such statements are related to the present context, so in the example and assuming reality under European legislation we would also have:

$sleep\_with\ COUNTS\ AS\ adultery$
$adultery\ COUNTS\ AS\ immoral$
$adultery\ COUNTS\ AS\ unethical$
$murder\ COUNTS\ AS\ unethical$
$adultery\ COUNTS\ AS\ illegal$

Clearly, we will also have general context-independent statement that we do not consider here. We now show self-checking constraints that usefully employ $COUNTS\ AS$ facts. Such facts are either explicit or can implicitly derived by transitivity (we do not enter in the detail of how to implement transitivity).

Below we introduce a constraint for context change:

$ALWAYS\ \ context\_change(C, C_1) \div$
$\quad discharge\_context(C),\ assume\_context(C_1)$

In particular, whenever the agent perceives a change of context (e.g., the agent stops working and starts a videogame, or finishes a videogame and goes to help children with their homework, etc.) then all the relevant ethic assumptions (among which, for instance, the $COUNTS\ AS$ facts) about the new context C_1 must be loaded, while those relative to previous context C must be dismissed; this is important because, e.g., after finishing a videogame it is no longer allowed to kill any living being in view just for fun... Frequency of check of this constraint is not specified here, however it should guarantee a prompt enough adaptation to a change.

Given now the present context for granted, no plan or single action can be allowed which counts as unethical in the context. So, we can have the following constraints:

$NEVER\ \ goal(G), plan(G, P), element(Action, P) ::$
$\quad Action\ COUNTS\ AS\ unethical \div execute\_plan(P)$

The next example is a meta-statement expressing the capability of an agent to modify its own behavior. If a goal G which is crucial to the agent for its ethical behavior (e.g., providing a doctor or an ambulance to a patient in need) has not been achieved (in a certain context) and the initially allotted time has elapsed, then the recovery component implies replacing the planning module (if more than one is available) and retrying the goal. We suppose that the possibility of achieving a goal G is evaluated w.r.t. a module M that represents the context for G (notation $P(G, M)$, P standing for 'possible'). Necessity and possibility evaluation with reasonable complexity by means of Answer Set Programming (ASP) modules has been proposed and discussed in [40][3]. If the goal is still deemed to be possible but has not been achieved before a certain deadline, the reaction consists in substituting the present planning module and re-trying the goal.

$NEVER\ goal(G),$
$\quad eval\_context(G, M), P(G, M),$
$\quad timed\_out(G), not\ achieved(G) \div$
$\qquad replace\_planning\_module, retry(G)$

Time intervals have never been exploited in the above examples. It can however been useful in many cases for the punctual definition of moral/ethical specific behaviors, e.g., never leave a patient or a child alone at night, and the like.

[3] ASP (cf., among many, [**?**,41–43] and the references therein) is a successful logic programming paradigm which is nowadays a state-of-the-art tool for planning and reasoning with affordable complexity, for which many efficient implementations are freely available [**?**].

6 Related Work and Concluding Remarks

In this paper we have proposed to adopt special metarules and runtime constraints for agents' self-checking and monitoring in the perspective of implementing 'humanized' agents. We have shown how to express useful properties apt to enforce ethical behavior in agents. We have provided a flexible framework, general enough to accommodate several logic-based agent-oriented languages, so as to allow both metarules and constraints to be adopted in different settings.

We may notice similarities with event-calculus formulations [44]. In fact, recent work presented in [45] extends the event calculus for a-priori checking of agents' plans. [46] treats the run-time checking of actions performed by BDI agents, and proposes an implementation under the JADE platform; this approach is related to ours, though the temporal aspects and the correction of violations are not present there.

Standard deontic logic (SDL) and its extensions ([47], [48]) are regarded as theories of 'ought-to-be' (or also 'ought-to-do'), thus they are certainly applicable to Ethics issues. 'Per se', deontic logics are not defined for agents. I.e., these logics are not originally targeted at formalizing the concept of actions being obligatory, permissible, or forbidden for an agent. Moreover, despite many desirable properties SDL and related approaches are problematic because of various paradoxes and limitations ([48],[49]). Concerning deontic logics targeted to agents and actions, and thus adequate for the formalization of Machine Ethics issues, a suitable semantics had been proposed by [50] and the corresponding axiomatization has been investigated by [51]. For a survey on deontic logics developments the reader may refer to [48]. Deontic logics have been used for building well-behaved ethical agents, like, e.g., in the approach of [52]. However, this approach requires an expressive deontic logic. To obtain such expressiveness (while of course not compromising efficiency), one needs highly hybrid modal and deontic logics that are undecidable. Even for decidable logics such as the zero-order version of Horty's System ([50]), decision procedures are likely to exhibit inordinate computational complexity. In addition, their approach is not generally applicable to agent-oriented frameworks. Therefore, although our approach cannot compete in expressiveness with deontic logic, still in its simplicity it can be usefully exploited in practical applications.

The approach proposed in this paper has been prototypically implemented using the DALI agent-oriented logic programming language, invented [53, 30] and implemented [54, 55] by our research group. DALI has a native construct, the *internal events feature*, which allows the implementation and proactive invocation of the proposed constraints. DALI is also equipped with modular capabilities and can invoke ASP modules. A more complete implementation and a proper experimentation will be the subject of forthcoming future work.

Future developments also include making self-checking constraints adaptable to changing conditions, thus to some extent emulating what humans would be able to do. This, as suggested in [7], might be done via automated synthesis of runtime constraints. This by extracting from the history of an agent's activity invariants expressing relevant situations. An important issue is that of devising a useful integration and synergy between declarative a-priori verification techniques such as those of [45] with the proposed run-time self-checking. The idea of [46] of a dynamic set of abstract and active rules will also be taken into serious consideration.

References

1. Bordini, R.H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A.E., Gómez-Sanz, J.J., Leite, J., O'Hare, G.M.P., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica (Slovenia) **30**(1) (2006) 33–44
2. Kouvaros, P., Lomuscio, A.: Verifying fault-tolerance in parameterised multi-agent systems. In Sierra, C., ed.: Proc. of the Twenty-Sixth Intl. Joint Conf. on Artificial Intelligence, IJCAI2017, ijcai.org (2017) 288–294
3. Shapiro, S., Lespérance, Y., Levesque, H.: The cognitive agents specification language and verification environment (2010)
4. Tørresen, J., Plessl, C., Yao, X.: Self-aware and self-expressive systems. IEEE Computer **48**(7) (2015) 18–20
5. Amir, E., Andreson, M.L., Chaudri, V.K.: Report on darpa workshop on self aware computer systems. Technical Report, SRI International Menlo Park United States (2007) Full Text : `http://www.dtic.mil/dtic/tr/fulltext/u2/1002393.pdf`.
6. Anderson, M.L., Perlis, D.: Logic, self-awareness and self-improvement: the metacognitive loop and the problem of brittleness. J. Log. Comput. **15**(1) (2005) 21–40
7. Rushby, J.M.: Runtime certification. In Leucker, M., ed.: Runtime Verification, 8th Intl. Works., RV 2008. Selected Papers. Volume 5289 of LNCS. Springer (2008) 21–35
8. Konolige, K.: Reasoning by introspection. In: Meta-Level Architectures and Reflection. North-Holland (1988) 61–74
9. van Harmelen, F., Wielinga, B., Bredeweg, B., Schreiber, G., Karbach, W., Reinders, M., Voss, A., Akkermans, H., Bartsch-Spörl, B., Vinkhuyzen, E.: Knowledge-level reflection. In: Enhancing the Knowledge Engineering Process – Contributions from ESPRIT. Elsevier Science (1992) 175–204
10. Perlis, D., Subrahmanian, V.S.: Meta-languages, reflection principles, and self-reference. In: Handbook of Logic in Artificial Intelligence and Logic Programming, Volume2, Deduction Methodologies. Oxford University Press (1994) 323–358
11. Barklund, J., Dell'Acqua, P., Costantini, S., Lanzarone, G.A.: Reflection principles in comp. logic. J. Log. Comput. **10**(6) (2000) 743–786
12. Costantini, S.: Meta-reasoning: a Survey. In: Comp. Logic: Logic Pr. and Beyond, Essays in Honour of Robert A. Kowalski, Part II. Volume 2408 of LNCS., Springer (2002) 253–288
13. Pereira, L.M., Saptawijaya, A.: Pr. Machine Ethics. Volume 26 of Studies in Applied Philosophy, Epistemology and Rational Ethics. Springer (2016)
14. Rao, A.S., Georgeff, M.: Modeling rational agents within a BDI-architecture. In: Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann (1991) 473–484
15. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP model of agency: Computational model and prototype implementation. In: Global Computing: IST/FET Intl. Works., Revised Selected Papers. LNAI 3267. Springer-Verlag, Berlin (2005) 340–367
16. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: Proc. ECAI-2004. (2004)
17. van Harmelen, F.: Definable naming relations in meta-level systems. In: Meta-Programming in Logic. LNCS 649, Berlin, Springer (1992) 89–104

18. Barklund, J., Costantini, S., Dell'Acqua, P., Lanzarone, G.A.: Semantical properties of encodings in logic programming. In: Logic Programming – Proc. 1995 Intl. Symp., Cambridge, Mass., MIT Press (1995) 288–302
19. Smith, B.C.: Reflection and semantics in lisp. In: Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages. (1984) 23–35
20. Dell'Acqua, P.: Development of an interpreter for a metalogic programming language. M.Sc. in Computer Science at the Dept. of Computer Science, Univ. degli Studi di Milano, Italy (1989) Supervisor Prof. Stefania Costantini, in Italian.
21. Costantini, S., Lanzarone, G.A.: A metalogic programming language. In: Logic Programming, Proceedings of the Sixth International Conference, MIT Press (1989) 218–233
22. Grosof, B.N., Kifer, M., Fodor, P.: Rulelog: Highly expressive semantic rules with scalable deep reasoning. In: Pr. of the Doctoral Consortium, Challenge, Industry Track, Tutorials and Posters @ RuleML+RR 2017 hosted by RuleML+RR 2017. Volume 1875 of CEUR Workshop Pr., CEUR-WS.org (2017)
23. Lloyd, J.W.: Foundations of Logic Programming, Second Edition. Springer, Berlin (1987)
24. Costantini, S., Lanzarone, G.A.: A metalogic programming approach: language, semantics and applications. J. Exp. Theor. Artif. Intell. **6**(3) (1994) 239–287
25. Costantini, S., Lanzarone, G.A.: Metalevel negation and non-monotonic reasoning. Meth. of Logic in CS **1**(1) (1994) 111
26. Dix, J.: A classification theory of semantics of normal logic programs: I. Strong properties. Fundam. Inform. **22**(3) (1995) 227–255
27. Rao, A.S.: Agentspeak(l): BDI agents speak out in a logical computable language. In: Agents Breaking Away, 7th European Works. on Modelling Autonomous Agents in a Multi-Agent World, Proceedings. Volume 1038 of LNCS., Springer (1996) 42–55
28. Hindriks, K.V.: Programming rational agents in goal. In: Multi-Agent Programming. Springer US (2009) 119–157
29. Dastani, M., van Riemsdijk, M.B., Meyer, J.J.C.: Pr. multi-agent systems in 3APL. In: Multi-Agent Programming: Languages, Platforms and Applications. Volume 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer (2005) 39–67
30. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In Alferes, J.J., Leite, J.A., eds.: Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Proceedings. Volume 3229 of Lecture Notes in Computer Science., Springer (2004) 685–688
31. Francalanza, A., Aceto, L., Achilleos, A., Attard, D.P., Cassar, I., Monica, D.D., Ingólfsdóttir, A.: A foundation for runtime monitoring. In: Runtime Verification - 17th International Conference, RV 2017, Proceedings. (2017) 8–29
32. Costantini, S., Dell'Acqua, P., Pereira, L.M.: A multi-layer framework for evolving and learning agents. In M. T. Cox, A.R., ed.: Proc. of Metareasoning: Thinking about thinking Works. at AAAI 2008, Chicago, USA. (2008)
33. Costantini, S.: Self-checking logical agents. In Osorio, M., Zepeda, C., Olmos, I., Carballido, J.L., Ramírez, R.C.M., eds.: Proceedings of the Eighth Latin American Workshop on Logic / Languages, Algorithms and New Methods of Reasoning 2012. Volume 911 of CEUR Workshop Proceedings., CEUR-WS.org (2012) 3–30 Extended Abstract in Proceedings of AAMAS2013.
34. Costantini, S., Gasperis, G.D.: Runtime self-checking via temporal (meta-)axioms for assurance of logical agent systems. In: Proc. of the 29th Italian Conf. on Comp. Logic CILC 2014. Volume 1195 of CEUR Works. Proc., CEUR-WS.org (2014) 241–255
35. Ben-Ari, M., Manna, Z., Pnueli, A.: The temporal logic of branching time. Acta Informatica **20** (1983) 207–226
36. Emerson, E.A.: Temporal and modal logic. In van Leeuwen, J., ed.: Handbook of Theoretical Comp. Sc., vol. B. MIT Press (1990)

37. Lichtenstein, O., Pnueli, A., Zuch, L.: The glory of the past. In: Proc. Conf. on Logics of Programs. LNCS 193, Springer Verlag (1985)
38. Cassar, I., Francalanza, A., Attard, D.P., Aceto, L., Ingólfsdóttir, A.: A suite of monitoring tools for erlang. In: RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools. (2017) 41–47
39. Costantini, S., Tocchio, A.: About declarative semantics of logic-based agent languages. In Baldoni, M., Endriss, U., Omicini, A., Torroni, P., eds.: Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Selected and Revised Papers. Volume 3904 of Lecture Notes in Computer Science., Springer (2005) 106–123
40. Costantini, S.: Answer set modules for logical agents. In: Datalog Reloaded: First Intl. Works., Datalog 2010. Volume 6702 of LNCS. Springer (2011) Revised selected papers.
41. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)
42. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In Baral, C., Brewka, G., Schlipf, J., eds.: Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007. (2007)
43. Truszczyński, M.: Logic programming for knowledge representation. In Dahl, V., Niemelä, I., eds.: Logic Programming, 23rd International Conference, ICLP 2007. (2007) 76–88
44. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing **4** (1986) 67–95
45. Berreby, F., Bourgne, G., Ganascia, J.: A declarative modular framework for representing and applying ethical principles. In Larson, K., Winikoff, M., Das, S., Durfee, E.H., eds.: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, ACM (2017) 96–104
46. Tufis, M., Ganascia, J.: A normative extension for the BDI ahent model. In: Proceedings of the 17th INternational Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines. (2014) 691–702
47. Åqvist, L.: Deontic logic. In: Handbook of philosophical logic. Springer (1984) 605–714
48. Hilpinen, R., McNamara, P.: Deontic logic: a historical survey and introduction. Handbook of deontic logic and normative systems. College Publications **80** (2013)
49. Broersen, J.M., van der Torre, L.W.N.: Ten problems of deontic logic and normative reasoning in computer science. In: ESSLLI. Volume 7388 of Lecture Notes in Computer Science., Springer (2011) 55–88
50. Horty, J.F.: Agency and deontic logic. Oxford University Press (2001)
51. Murakami, Y.: Utilitarian deontic logic. In: Advances in Modal Logic, King's College Publications (2004) 211–230
52. Bringsjord, S., Arkoudas, K., Bello, P.: Toward a general logicist methodology for engineering ethically correct robots. IEEE Intelligent Systems **21**(4) (2006) 38–44
53. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In Flesca, S., Greco, S., Leone, N., Ianni, G., eds.: Logics in Artificial Intelligence, European Conference, JELIA 2002, Proceedings. Volume 2424 of Lecture Notes in Computer Science., Springer (2002)
54. De Gasperis, G., Costantini, S., Nazzicone, G.: Dali multi agent systems framework, doi 10.5281/zenodo.11042. DALI GitHub Software Repository (July 2014) DALI: `http://github.com/AAAI-DISIM-UnivAQ/DALI`.
55. Costantini, S., De Gasperis, G., Pitoni, V., Salutari, A.: Dali: A multi agent system framework for the web, cognitive robotic and complex event processing. In: Proceedings of the 32nd Italian Conference on Computational Logic. Volume 1949 of CEUR Workshop Proceedings., CEUR-WS.org (2017) 286–300 http://ceur-ws.org/Vol-1949/CILCpaper05.pdf.