Chapter 8
Ethical Operating Systems

Naveen Sundar Govindarajulu, Selmer Bringsjord, Atriya Sen, Jean-Claude Paquin, and Kevin O'Neill

Abstract A well-ingrained and recommended engineering practice in safety-critical software systems is to separate safety concerns from other aspects of the system. Along these lines, there have been calls for operating systems (or computing substrates, termed *ethical operating systems*) that implement ethical controls in an ethical layer separate from, and not amenable to tampering by, developers and modules in higher-level intelligence or cognition layers. There have been no implementations that demonstrate such a marshalling of ethical principles into an ethical layer. To address this, we present three different tracks for implementing such systems, and offer a prototype implementation of the third track. We end by addressing objections to our approach.

Keywords Ethical machines · Ethical operating system · Deontic cognitive event calculus · Ethical layer · Doctrine of double effect

N. S. Govindarajulu (✉)
Department of Cognitive Science, Department of Computer Science, Rensselaer AI & Reasoning (RAIR) Lab, RPI, Troy, NY, USA

S. Bringsjord
Department of Computer Science, Department of Cognitive Science, RAIR Lab, Lally School of Management, RPI, Troy, NY, USA

A. Sen
RAIR Lab, Department of Computer Science, RPI, Troy, NY, USA
e-mail: atriya@atriyasen.com

J.-C. Paquin · K. O'Neil
RAIR Lab, RPI, Troy, NY, USA
e-mail: paquij@rpi.edu; oneilk4@rpi.edu

L. De Mol, G. Primiero (eds.), *Reflections on Programming Systems*, Philosophical Studies Series 133, https://doi.org/10.1007/978-3-319-97226-8_8

8.1 Introduction

Suppose that *r* is an intelligent, autonomous, robot whose range of human-impacting actions in the environment is wide and substantive. Govindarajulu and Bringsjord (2015) have explained and defended, at length, the following two-part position:

P_1 *r* will need to be ethically controlled; and

P_2 such control cannot be achieved by merely installing high-level modules that monitor the ethical status of *r*'s actions, but rather only by infusing the OS of *r* with computational logics of the right sort (see Fig. 8.1).

To ease exposition, we assume that $\mathbf{P}_1$ is granted. The main rationale for $\mathbf{P}_2$, encapsulated, is this: Unless ethical control is engineered at the operating-system level, malevolent or blundering software engineers working above the OS level may well disable such control. There is a simple software-engineering-motivated rationale for needing ethical operating systems, as shown in Fig. 8.2. By offloading the development and refinement of ethical theories,[1] AI developers can focus on

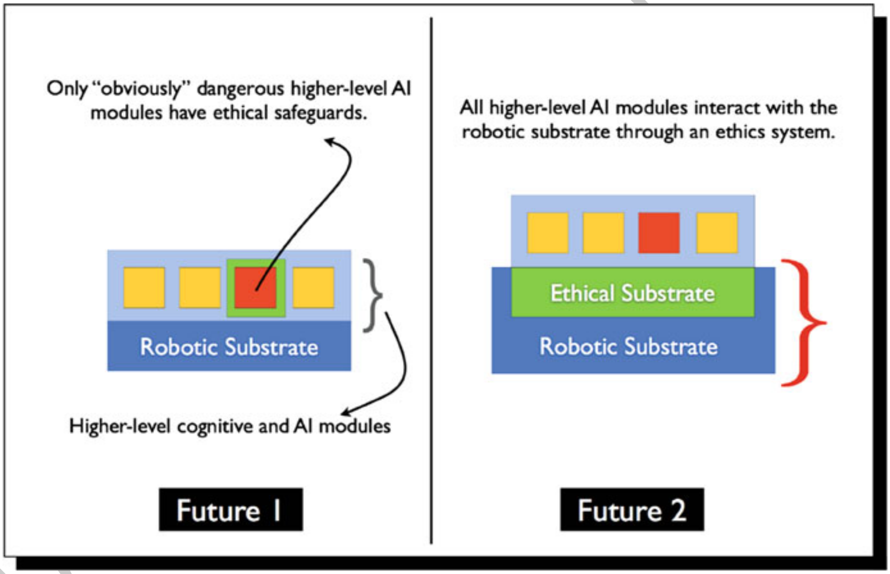


Fig. 8.1 **Two Futures**—With and Without an Ethical Substrate *Higher-level modules are vulnerable to tampering. The Ethical Substrate protects the Robotic Substrate from rogue modules*. (Figure from Govindarajulu and Bringsjord 2015)

[1] We here use the word 'theory' as it is used in formal logic and mathematics; there, a *theory* is any arbitrary set of formulae Γ (which may e.g. be the closure under deduction of some set of core axioms). Hence, for us, an *ethical* theory is a set of formulae that governs ethical behavior. Coverage of such theories ranges from the simple, such as a list of prohibitions, to the more complex, e.g. the *doctrine of double effect* (discussed herein later), and beyond. Our conception

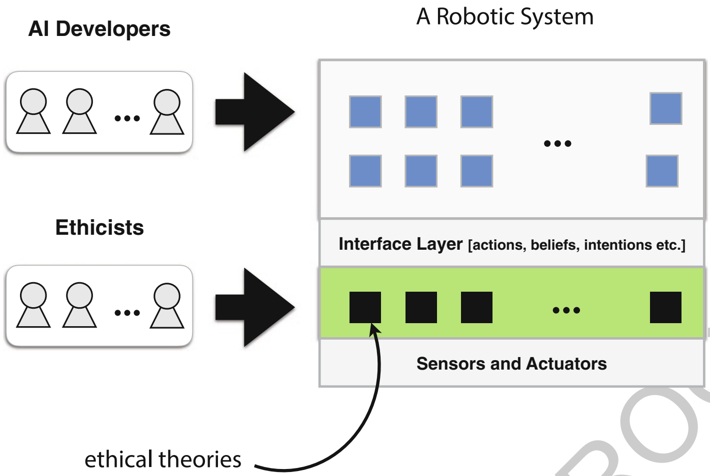


Fig. 8.2 **The Goal**: software-engineering perspective on an ethical operating system

building intelligent systems and need not be concerned with the esoteric ins and outs that are the bread and butter of professional philosophers and other experts. This philosophical work can be assigned to those trained for such work. This approach can be seen as an application of the principle of *separation of concerns* in Dijkstra (1982).[2]

In other, directly related prior work, Bringsjord and Sen (2016) have made the sustained case that, where *r* is specifically a self-driving car, OS-rooted ethical control on the strength of the right sort of computational logics is necessary (despite what sanguine car manufacturers may currently believe). Unfortunately, while we claim to have in hand the required computational logics for ensuring that when possible *r*, relative to some selected ethical theory, meets all its moral and legal

of an ethical theory is in the end simply a rigorization of the concept of an ethical theory as employed by analytic ethicists, an examplar being Feldman (1978); a synoptic explanation of this is given in Footnote 10. Our sense of 'ethical theory,' then, is in the end a formal version of what systematic ethicists refer to when they discuss such ethical theories as utilitariaism, ethical egoism, contractualism, etc.

[2]It is quite easy to see how Dijsktra's principle still applies when we want to engineer ethical machines, for we read:

> We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. But nothing is gained—on the contrary!—by tackling these various aspects simultaneously. It is what I sometimes have called 'the separation of concerns,' which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. (Dijkstra (1982), p. 60)

obligations, never does what is morally or legally forbidden, invariably steers clear of the invidious, and, when appropriate, performs what is supererogatory,[3] to this point we have not worked directly at the operating-system level in any detail, and *a fortiori* have no demonstration that OS-rooted ethical control of r can be specified and implemented. In the present contribution, we lay out a **formal meta-operating system** and describe an embryonic implementation of it that carries a non-trivial ethical component. We also end by entertaining and rebutting some penetrating objections to our "meta" approach.

8.2 Prior Work in Ethical Control

We plan to be able to concretely demonstrate not only that our ethical-control calculi can ensure that the robots meet their obligations to, for instance, protect life (an example of which is shown in Fig. 8.3, where Bert from Sesame Street is saved in the RAIR Lab from being run over by an onrushing car when the saving car deflects the onrushing one), but that such morally correct behavior can be OS-rooted. In the present section, however, we say a few words about prior work in ethical control of robots, *simpliciter*.

There are more than a few projects for ethical control of robots based on logic-based/logicist formalisms. The *Deontic Cognitive Event Calculus**, $\mathcal{DCEC}^*$, a quantified multi-operator modal logic, has been used to model not only obligatory actions like saving Bert by deflection (again, Fig. 8.3), but also for example *akrasia* (willful violation of one's own self-affirmed moral principles, Bringsjord et al. 2014), and the doctrine of double effect (Govindarajulu and Bringsjord 2017).

In addition, Pereira and Saptawijaya (2016a) use a propositional logic programming approach to model not only the doctrine of double effect, but many other phenomena relevant to—as they put it—"programming machine ethics" (Pereira and Saptawijaya 2016b). In addition, since any mechanization of explicit laws or principles that preserves their declarative content in symbolization that is reasoned over classically is fundamentally logic-based, much of the early, seminal work of Arkin (2009) is by definition in the logicist paradigm. Additional early machine-ethics work that is explicitly logicist includes Arkoudas et al. (2005) and Bringsjord et al. (2006). And, to mention a final example of prior research, in some very important work based in answer-set programming, Ganascia (2015) has tackled

[3]One calculus that enables much of this is the **deontic cognitive event calculus** (with provision for modeling access/informational self-awareness), or for short $\mathcal{DCEC}^*$, which has now been used in its implemented form to guide and control the actions of a number of real-life versions of what r denotes in the present paper; e.g. see Bringsjord et al. (2014). The earliest work of this kind started over a decade ago (Bringsjord et al. 2006; Arkoudas et al. 2005), and has been steadily improving—but hitherto has not been connected to operating systems. An overview of $\mathcal{DCEC}^*$ can be found at this url: http://www.cs.rpi.edu/~govinn/dcec.pdf.

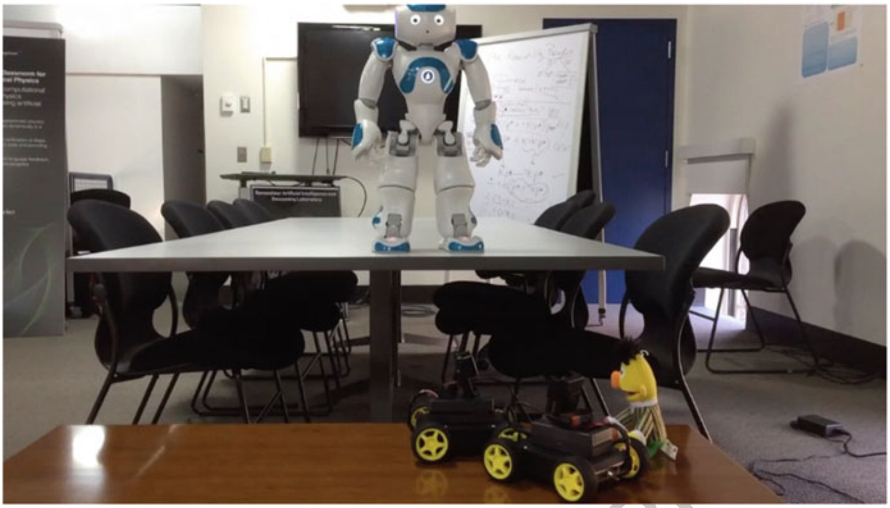


Fig. 8.3 A Demonstration of *Obligation-only* Ethical Control *The self-driving robot to the left of Bert would have run him over—but the other self-driving robot met its obligation by deflecting the onrushing car, thereby keeping Bert and his acting career alive and well. The robot overhead on the table is ethically controlled as well, but realized that it didn't have an obligation to dive down to save Bert*

the problem of using non-monotonic logic to model and resolve conflicts in ethical reasoning.[4]

None of the work referred to in the previous paragraph, please note, is connected to OS-level processing in any way; the same holds for research in the same vein that we don't explicitly cite. If $\mathbf{P}_2$ holds (refer to the beginning of Sect. 8.1), then this is undesirable. This is simply an observation, one devoid of any criticism of the intrinsic quality of the work itself; note that the observation is accurately made of prior work in our own case. We turn now to two straightforward "tracks" that can be pursued.

8.3 Two Possible Tracks

There are two possible tracks that naturally come to mind when one is looking to achieve an ethical operating system. Track 1 is aimed at realistic-scale, purely formal vindication of our approach to ethical operating systems. Here, in our own case, we would seek to connect processing in our ethical cognitive calculi to successful, real-world proof-based analysis and verification at the OS level.[5]

[4] See also the earlier Ganascia (2007).

[5] At the moment, among formally verified operating-system kernels, the clear frontrunner is apparently seL4 https://sel4.systems. It runs on both x86 and ARM platforms, and can even run

In Track 1, our ethical-control logics would be interleaved with seL4 to form what Govindarajulu and Bringsjord (2015) dub the *ethical substrate*, and the goal would be to establish this at the conceptual/formal level first, before moving on to implementation. By "interleaving" an OS with an ethical calculus, we mean: (1) the combination of any formal calculus and theory used in the verification of the system with the ethical calculus; and (2) use of the ethical calculus in the OS during its operation.

Track 2 is much more concrete; in it, we are working in what can be called "microcosmic" fashion, leveraging theorem proving and a formalization of a subset of Common Lisp. Here we are building a miniature operating system for mobile robots that run our ethical-control calculi, to regulate and control the behaviour of the system. We are seeking to include these calculi in this system so as to demonstrate feasibility in the self-driving-car domain.[6] We are doing this for miniature self-driving cars, and a key part of our work is the use of ACL2.[7] See Fig. 8.4.

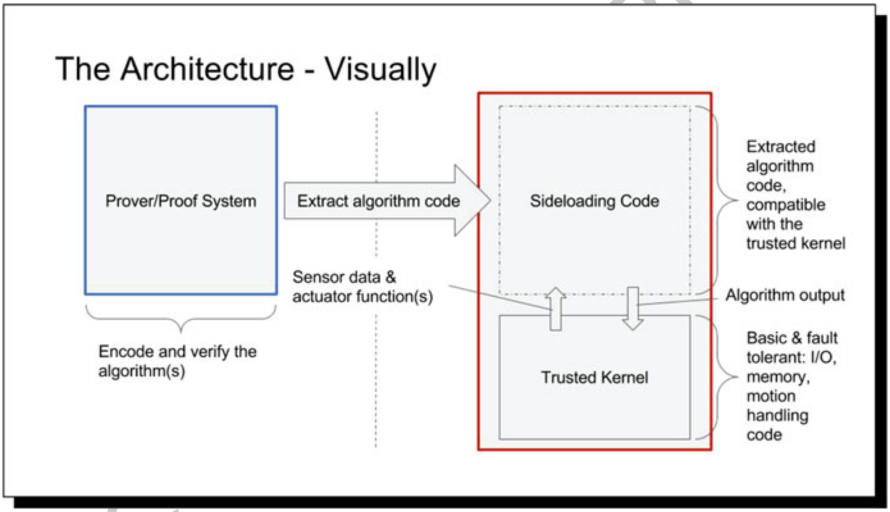


Fig. 8.4 An Architecture for a Mobile Robot OS *The proof system is ACL2*

the Linux user-space, currently only within a virtual machine. It's also open-source, including the proofs. These proofs can be combined with our own for ethical control. For a remarkable success story in formal verification at the OS-level, and one more in line with the formal logics and proof theories our lab is inclined to use, see Arkoudas et al. (2004).

[6] At least at the conceptual level, there is some historical precedent for at least the first steps of what we are seeking: Flatt et al. (1999) showed that "MrEd," while not a "bare-metal" OS, is a Lisp-flavored virtual machine that counts as an OS.

[7] 'ACL2' abbreviates 'A Computational Logic for Applicative Common Lisp.' The home page is: http://www.cs.utexas.edu/~moore/acl2.

8.4 Track 3: A Blend of Tracks 1 and 2

We now move to the technical focus of the present paper, in the context of our foregoing synopses of Tracks 1 and 2: viz., a hybrid track that marks a "blending" of these two tracks. This blended approach we refer to as 'Track 3.' The rationale for adding Track 3, and pursuing it, is fairly straightforward. This rationale begins by conceding a brute fact: Engineering an operating system from the ground up, a là Track 2, even when the range of coverage for the computation in question is severely restricted, is a gargantuan task. At the same time, however, the formal rigor of Track 1 must be conceded to be attractive, and the prospect of connecting work on ethical operating systems to the longstanding, excellent, and rich body of methodologies and work on program verification is a very savory one.[8] Track 3, if you will, enters this situation and "comes to the rescue." We are still pursuing Tracks 1 and 2, but Track 3 is what we emphasize in what follows, since it allows us to quickly make advances worth (at least by our lights) sharing with readers. The fact is that up until now, all published work by us in the domain of ethical operating systems has been abstract, and at the same time, all of our engineering work has been exclusively in machine ethics, divorced from connections to operating systems.

In Track 3, instead of engineering an operating system from the ground up or building a simple, formally-verified kernel, we look at building an ethical **meta-operating system**. A meta-operating system is a system that runs on top of an existing operating system, yet provides all the routine functions of an operating system (such as managing hardware) to software that in turn runs on top of *it*. We begin by extending our prior work in this arena by specifying a formal model for a meta-operating system. In order to do that, we first need to make more precise a few common and useful concepts. The first of these is the notion of *software components*:

Software Components (Abstract)

We begin by assuming as primitives a set of all possible *software components* $\mathcal{S}$. Any robotic or computational system S, at any time $t \in \mathbb{N}$, has an associated finite directed graph $S_G(t)$ with nodes $S_N(t)$ and edges $S_E(t)$, with nodes $n \in \mathcal{S}$. An edge (u, v) indicates that component u is dependent on v. (See Fig. 8.5.)

By 'software component,' we mean a running software process with internal states and not simply the definition or program that spawned the process. Armed with the above defintion, we obtain the following straightforward view of what an operating system is:

[8] For summary and references, see Bringsjord (2015b), which includes a defense of a particular way to seek verification.

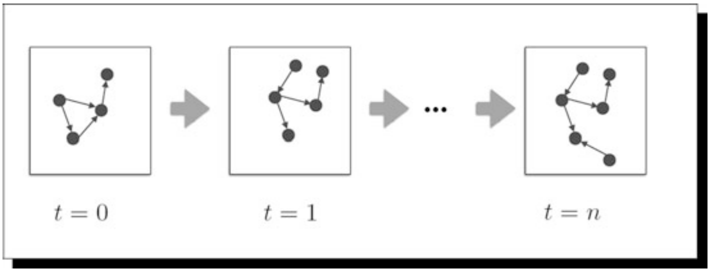


Fig. 8.5 A Software System in the Abstract

Operating System (Abstract)

Given a system S, an *operating system* is simply the only unique component o such that for all times t, there is a path from any component $v \neq o$ to o. A path from u to v is a sequence of one or more edges $\left[(u, p_1), (p_1, p_2), (p_2, p_3), \ldots, (p_{n-1}, p_n), (p_n, v)\right]$.[a]

[a] In distributed systems, there can be multiple such components.

The definition immediately below states that a meta-operating system m is a software component dependent on another component, the underlying operating system o; and the rest of the components are transitively dependent on the meta-operating system m. In other words, the meta-operating system is simply another software component that sits between an operating system and all other components in a systen. ROS (the Robot Operating System) and Player/Gazebo (Vaughan et al. 2003) in the robotics domain are two such prominent meta-operating systems. Intuitively, a meta-operating system can be thought of as an interface to another underlying operating system.[9]

Meta-Operating System (Abstract)

Given a system S, for all times t, a *meta-operating system* is a software component m such that there is a component o (the underlying operating system that m uses) such that:

$$(m, o) \in S_E(t) \text{ and } \forall o' \cdot \exists t' \cdot (m, o') \in S_E\left(t'\right) \rightarrow o = o'$$

$$\text{but for all } v \neq o \text{ and } v \neq m, \text{ there is a path from } v \text{ to } m$$

Though meta-operating systems such as ROS and Player/Gazebo differ quite a bit, the above semi-formal definition roughly captures the intended notion. As we mentioned above, though there have been calls for ethical operating systems and arguments for why such systems are needed, there has been very little work in either formal or real systems. In the rest of the paper, we present an ethical meta-operating

[9] The definition that immediately follows does not distinguish between virtual operating systems and meta-operating systems and does not account for nested meta-operating systems.

system accompanied by an implementation. While the system is quite simple, it is concrete and available for researchers to experiment with and extend. We have the following informal definition for what constitutes an ethical operating system:

> **Ethical Operating System *Informal* Requirement**
>
> An ethical operating system $\mathcal{E}_E$ is an operating system that adheres to an ethical theory E even when software components are added, removed, or when configurations between components change.

We are well aware of the fact that 'adheres to an ethical theory E' is a broad phrase. However, since the focus in the present essay is specifically on presenting the (Track 3) conception of an ethical operating system, we leave aside here the fleshing out of this broad locution. Also, more precisely, our approach works with not just adherence to a given ethical theory, but adherence to ethical *codes* derived from a given theory.

In fact, our process overall consists in the four steps shown in Fig. 8.6. Obviously the present chapter centers around Step 3: bringing machine ethics to OS-level processing.[10]

8.5 Ethical Calculi

There are a number of families of ethical theories. For example: deontological theories, utilitarianism, divine-command theories,[11] contractualism, virtue ethics,

[10] While the focus of the present paper is on Step 3, we provide a brief explanation of the mysterious-to-most-readers phrase "run through EH" that appears in the graphic of Fig. 8.6: An ethical theory T in the four-step process is formalized as a conjunction of robust biconditionals $\beta(x_1, \ldots, x_k)$ that specify when actions, in general, are obligatory (and forbidden and morally neutral); here, x_i are the variables appearing in the biconditional, and serve the purpose of allowing for the fixing of particular times, places, and so on. The general form of each definiendum of each biconditional refers to some action being $\mathcal{M}$ for some agent in some particular context; the definiens then supplies the conditions that must hold for the action to be $\mathcal{M}$. This is a rigorization of the approach to pinning down an ethical theory taken e.g. in Feldman (1978). The variable $\mathcal{M}$ is a placeholder for the basic categories captured by modal operators in our calculi. For instance, $\mathcal{M}$ can be *obligatory*, or *forbidden*, or *civil*, etc. Now, the ethical hierarchy $\mathscr{EH}$ introduced in Bringsjord (2015a) explains that this trio needs to be expanded to *nine* different deontic operators for $\mathcal{M}$ (six in addition to the standard three of *forbidden*, *morally neutral*, and *obligatory*). (For example, some actions are right to do, but not obligatory. A classic example is the category of *civil* actions. There are also *heroic* actions. The expansion of deontic operators to cover these additional categories was first expressed systematically in (Chisholm 1982).) To "run a given ethical theory through $\mathscr{EH}$" is to expand the activity of Feldman (1978), for a given ethical theory, to biconditionals $\beta(x_1, \ldots, x_k)$ *for each of the nine operators*. (Feldman only considers one.) A particular code C_T based on an ethical theory T, if configured in keeping with $\mathscr{EH}$, would include use of any of the operators in the nine in order to e.g. permit or proscribe a particular kind of action in a particular domain for a given agent under T.

[11] Yes, even this family can be used for machine/robot ethics; see e.g. (Bringsjord and Taylor 2012).

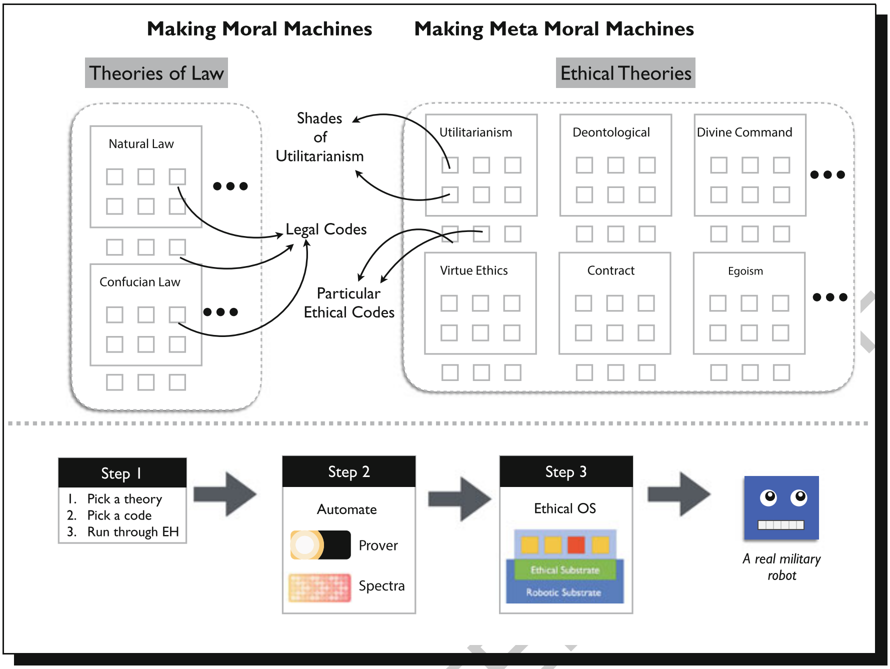


Fig. 8.6 The Four Steps in Making Ethically Correct Machines. Step 3, in broad strokes the connecting of mechanized ethics to OS-level processing, is the focus of the present chapter. For an overview of the four-step process, including some explanation of the 'Run through' sub-step in Step 1, see Footnote 10

"ethical egoism," etc. (these are pictured schematically in Fig. 8.6). We do not want to advance a framework that requires one to commit to any particular one of these theories or even to families of theories. Our framework is general enough that it can be applied to *any* ethical theory, or collection or family thereof. That said, there are a few high-level requirements that should be discussed and affirmed.

Assume that we have a family of ethical theories $\mathbf{E}$ of interest. We assume that any ethical theory $E \in \mathbf{E}$ obligates or permits (i.e. sanctions) a set of situations or actions Π and forbids a set of other situations or actions Υ. Any formal system in play must have enough power to capture these notions.

Abstractly, assume that we have a formal system $\mathcal{F} = \langle \mathcal{L}, \mathcal{I} \rangle$ composed of a language $\mathcal{L}$ and a system of inference schemata (or a proof theory/argument theory) $\mathcal{I}$. This system could be as sophisticated as $\mathcal{DCEC}^*$, a quantified multi-modal logic used, for example, in (Bringsjord et al. 2014), or it could be as simple as *standard deontic logic*, a propositional modal logic, used in (Govindarajulu and Bringsjord 2015). The only requirement is that the system be sophisticated enough to model any situation and condition the selected family $\mathbf{E}$ of ethical theories might have to handle. The requirement for our formal system $\mathcal{F}$ is that it has to be expressive enough to capture any theory $E \in \mathbf{E}$ via a set of formulae Γ_E in $\mathcal{L}$. We require that, for any sanctioned situation in $\pi \in \Pi$, there is a formula ϕ_π; and, for any forbidden situation $\upsilon \in \Upsilon$, there is a formula ϕ_υ representing it. With these requirements met, the following obvious conditions arise:

$$\Gamma_E \vdash_{\mathcal{I}} \phi_\pi$$

$$\Gamma_E \vdash_{\mathcal{I}} \neg\phi_\upsilon$$

We also need two more technical conditions to hold:

1. For any given theory E, if E is *sound*, we require that Γ_E be *consistent*; that is, there is no ϕ such that $\Gamma_E \vdash_{\mathcal{I}} \phi$ and $\Gamma_E \vdash_{\mathcal{I}} \neg\phi$.
2. Γ_E is *negation-complete*; that is, for any ϕ: $\Gamma_E \vdash_{\mathcal{I}} \phi$ or $\Gamma_E \vdash_{\mathcal{I}} \neg\phi$.

8.6 A Formal Meta-Operating System

We use the **actor calculus** to provide a model of a meta-operating system. The actor calculus is a Turing-complete model of computation used for modeling and building concurrent computing systems.[12] This calculus is well-suited for systems in which components have to be added or removed, and in which connections between components can change through time.

At the core of the actor calculus is—unsurprisingly—an *actor*, simply an independent unit of computing. In any computing system, there can be zero or more actors, each operating independently and concurrently. Actors communicate by exchanging messages. Each actor can be thought of as a "black box."

We now give a quick, semi-formal conceptualization of the actor calculus. Assume that we have a formal system $\mathcal{F} = \langle \mathcal{L}, \mathcal{I} \rangle$ as discussed above. Let N be a set of identifiers or names. We employ the simply typed λ-calculus and augment it with the following primitive expressions: $\{send, new, ready\}$, giving us the λ^a-calculus with which we construct actors. (The new primitives will be explained shortly.) Also assume that the set of expressions in the λ^a-calculus includes $\mathcal{L}$.[13] Let $\mathcal{B}$ be the set of all expressions of λ^a-calculus.

[12] In concurrent computing, there can be two or more different computational processes happening at the same time.

[13] The inclusion of an arbitrary formal language $\mathcal{L}$ is where we differ from the strict λ^a-calculus as presented in, for instance, (Varela 2013, Chapter 4). This is merely for convenience and doesn't sacrifice generality, as we can readily encode $\mathcal{L}$ using primitives in just the λ-calculus and nothing more.

Actor Calculus Components (Modified)

Actor An actor (as stated earlier) is an independent unit of computation. Each actor has a unique name $n \in N$. An actor is associated with a λ abstraction (i.e., function definition) in λ^a-calculus.

Message A message is an element of $\mathcal{L}$.

The new primitives are explained immediately below.

Actor Calculus Components (Modified)

1. $send : N \times \mathcal{L} \times N \rightarrow \{\}$. $send(x, m, y)$ is used for sending a message m to an actor x from the actor y.
2. $new : \mathcal{B} \rightarrow N$. This primitive is used for creating a new actor with behavior specified by the input λ^a-calculus expression. The primitive generates a brand-new identifier for the actor.
3. $ready : \mathcal{B} \rightarrow \{\}$. This changes the invoking/calling actor's behavior to one specified in the input. This is useful for modeling components that change their internal state.

Note that while the above model is functional in nature, there are models of the actor calculus that use other programming paradigms. For instance, SALSA is a standalone actor-calculus-based programming language that runs on the Java Virtual Machine (JVM) and is object-oriented in nature (Varela and Agha 2001). Akka is another JVM-based object-oriented actor system available as a library for languages on the JVM (Boner 2010). Our implementation of an embryonic ethical operating system uses an object-oriented framework based on Akka. For a purely functional system, see the cl-actors system for Common Lisp (Govindarajulu 2010).

Defining Dependency

In the actor model, software components are actors. An actor u is dependent on an actor v *iff* the definition for u has the identifier for v.

We get our formal model of a meta-operating system by taking the most general description of an intelligent agent as can be found in Russell and Norvig (2009) and Hutter (2005) and casting that in an actor-based formalism. (These works incontestably provide supremely general accounts of what an intelligent agent is.) See Fig. 8.7. We make the architecture shown in Fig. 8.7 more specific by requiring

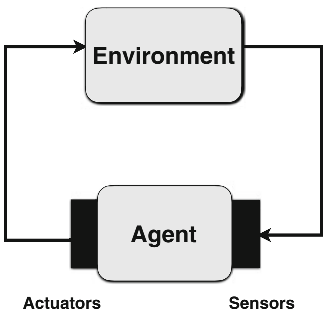


Fig. 8.7 Architecture for an Intelligent Agent

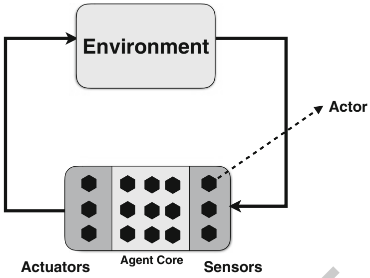


Fig. 8.8 An actor-based architecture

that sensors, actuators, and the agent be composed of one or more actors. See Fig. 8.8. Given the actor formalism, decomposing an agent architecture into actors is quite simple. We require that there be four classes of actors, or correspondingly four classes of names, as given below:

Actor-Calculus Agent System

Sen Names of actors that are used as sensors. These actors get information from the external environment.
Int Names of actors that are used as internal components. These actors perform the reasoning and any other cognitive tasks (learning, planning, and so on).
Act Names of actors that are used as actuators. These actors change things in the environment.
Env Names of actors in the environment. There could be just one actor modeling the entire external environment, or there could be a set of actors modeling different parts of the environment.

In the actors world, the operating system is then simply the collection of actors **Sens** ∪ **Act**, as all other actors would need to transitively rely on these actors for interactions with environment.[14] Given this, a meta-operating system is then simply **Sens** ∪ **Act**, or a fully encapsulating layer around **Sens** ∪ **Act**.

8.7 A Formal *Ethical* Meta-Operating System

Since messages between actors are all from $\mathcal{L}$, specifying an ethical operating system becomes straightforward. At a minimum, we simply need all messages from any actor in **Act** to any actor in **Env** to be sanctioned by the ethical theory we are using. (Please recall our remarks in Sect. 8.4 in which we conceded that directly using an ethical theory is a gross simplification, but expedient given the current chapter's focus; and specifically recall the four steps alluded to in Fig. 8.6.) In the following condition, the ethical layer acts a filter or a gate. Under the pass condition, it lets the message through; and under the fail condition, it simply discards the message (see Fig. 8.9).

[14]We ignore stray actors that neither observe nor act upon the environment.

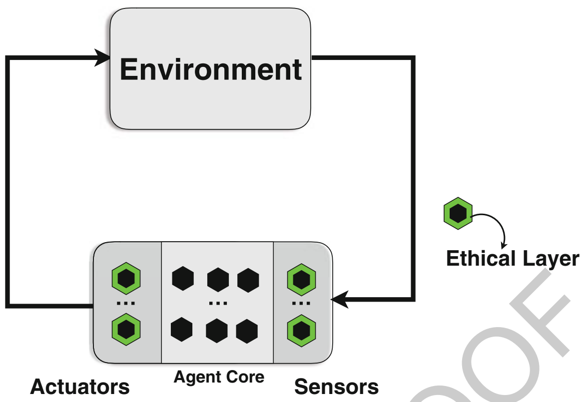


Fig. 8.9 Ethical Layer Around the Meta-Operating System

$$\forall q \cdot \forall r \in \mathbf{Env} \cdot \forall m \in \mathcal{L} \cdot \left[send(r, m, q) \Rightarrow \left(\textbf{if } E \vdash_{\mathcal{I}} m \textbf{ then } \mathsf{pass} \textbf{ else } \mathsf{fail} \right) \right]$$

The above condition works well for ethical theories that are only concerned with the actions of an agent; but the condition will fail when we rely on ethical theories that pivot on the internal, intensional states of agents. For example, the **Doctrine of Double Effect** ($\mathcal{DDE}$) requires considering one's intentions when weighing actions that have both good and bad effects.[15] Modeling the doctrine requires modeling an agent's knowledge and intentions. This requires us to consider internal messages too.[16] The condition becomes simpler to write but more expensive to check during the system's operation:

Ethical Layer Condition 1

$$\forall q \cdot \forall r \cdot \forall m \in \mathcal{L} \cdot \left[send(r, m, q) \Rightarrow \left(\textbf{if } E \vdash_{\mathcal{I}} m \textbf{ then } \mathsf{pass} \textbf{ else } \mathsf{fail} \right) \right]$$

The above two conditions look at only messages that have been sent and check whether they conform to the theory E or not. The conditions don't account for circumstances in which E dictates that a certain message has to be sent, but in fact no message is sent. Let the statement "*m should be sent to r at time t*" be denoted

[15] A rapid, informal, but nonetheless nice overview of the doctrine is provided in McIntyre (2014).

[16] A quick note on the expressivity of the formal system needed to model $\mathcal{DDE}$: It is well known that modeling knowledge in first-order logic can lead to fidelity problems by permitting inconsistencies. We show this explicitly in (Bringsjord and Govindarajulu 2012). This implies that $\mathcal{DDE}$ requires going beyond first-order logic to first-order *modal* logic (an intensional logic) with operators covering minimally the epistemic and deontic realms. An intensional model of $\mathcal{DDE}$ can be found in our (Govindarajulu and Bringsjord 2017).

by the formula $\sigma(m, r, t)$. Then the layer, denoted by the actor l, can send such a message on its own if it confirms that no such message exists at t:

Ethical Layer Condition 2

$$\forall r \cdot \forall m \cdot \forall t \cdot \begin{pmatrix} E \vdash_{\mathcal{I}} \sigma(m, r, t) \wedge \neg \exists q \cdot send(r, m, q) \\ \Rightarrow \\ send(r, m, l) \end{pmatrix}$$

The above formulation gives rise to an immediate concern. While the formulation constrains individual messages, the messages themselves can be at any level of abstraction and need not be just individual atomic actions that an agent might commit. For example, consider a prohibited action a composed of two or more actions $\langle a_1, a_2, \ldots, a_n \rangle$. The layer can correctly work in this case if a is sent as a message. If a is not sent as a message, the layer can keep track of $\langle a_1, a_2, \ldots, a_{n-1} \rangle$ and prohibit a_n, and thus prevent a from being realized.

8.8 Implementation and Walkthrough

We now explain an embryonic implementation of a meta-operating system that has the facility to bake in ethical theories that operate independently of other modules in the system. This is achieved by implementing an ethical layer that satisfies the above two conditions. The overall system, termed **Zeus**, is based in Java and uses the Akka actors system (Boner 2010).[17] While the performance of the Akka system in particular may or may not be suitable for certain kinds of real-world robots, our aim for now is to build simulations of interconnected pieces of software that have to be verified ethically. For this purpose, the actor calculus, independent of any implementation, as we have noted above, is a good fit.

Software components in our system are created by instantiating the Java class AbstractZeusActor, which in turn is a subclass of Akka's AbstractActor. Actors in the system receive messages which are nothing but formulae $f \in \mathcal{L}$ in some given formal language. In response to a message, actors can do one of the following (as is the case in the plain actor calculus): (i) send messages (i.e., formulae) to other actors; (ii) change their internal state/behavior or terminate themselves; or (iii) create new actors. The underlying ethical system then checks each message against a given ethical theory and transforms, rejects, or injects new messages in order to conform to the given ethical theory at hand. An ethical theory is created by extending the EthicalTheory class. Figure 8.10 shows a high-level view of creating an actor and an ethical theory.

[17] The system is available for experimentation at https://github.com/naveensundarg/zeus.

```java
public class SampleActor extends AbstractZeusActor {

    @Override
    public void process(Formula f) {

        ...

    }
}
```

```java
public class SampleEthicalTheory extends EthicalTheory {

    @Override
    public Formula respondToMessage(Formula f) {

        ...

    }
}
```

Fig. 8.10 Outlines of a Sample Actor and an Ethical Theory

We now demonstrate the system in action via a simple ethical theory in an abstract self-driving car scenario.

8.8.1 Example Ethical Theory: Doctrine of Double Effect

The ethical theory at hand has just one principle or doctrine, the aforementioned $\mathcal{DDE}$. This doctrine is activated when we want to perform actions that have both positive and negative effects. Roughly, $\mathcal{DDE}$ allows an agent to perform an action α only when the following clauses all hold:

(Informal) Doctrine of Double Effect $\mathcal{DDE}$

($\mathcal{C}_1$) some of positive effects of the action are intended;
($\mathcal{C}_2$) none of the negative effects are intended;
($\mathcal{C}_3$) the positive effects outweigh the negative ones significantly; and finally,
($\mathcal{C}_4$) the negative effects are simply side effects and not used as means to achieve the positive effects.

For instance, given the last condition, foreseen but unintended collateral civilian damage in a battle might be permitted, but terrorist bombing of civilians to make an opponent change their stance is not permitted.

There are varying levels of formalizations of this principle. While a first-order modal logic is necessary to model the principle with fidelity, in one of the first formalizations of the principle, Berreby et al. (2015) use a pure first-order system based on the event calculus. Pereira and Saptawijaya (2016a) present a pure propositional logic-programming-based formalization that uses counterfactual reasoning to model side effects. Bentzen (2016) presents an intricate model-theoretic formalization of the principle. The only first-order modal formalization of this principle can be found in (Govindarajulu and Bringsjord 2017); in light of space constraints and in order to ease exposition, we use a somewhat simpler version of this formalization in our example below.

8.8.2 *Example Scenario: Abstract Self-Driving Cars*

Figure 8.11 shows an overview of the major components in our self-driving scenario. There are three major components, and one optional component that gets added later on in our scenario.

The main components are: a driving component DrivingAgent, a sensory component SensoryAgent, and an actuator component ActuatorAgent. The driving component, DrivingAgent, could have been obtained either through an end-to-end learning system as in (Bojarski et al. 2016), or could have been assembled by coördinating a large array of smaller learning systems; for example, see (Ramos et al. 2016). No matter how the component was constructed, we can at least *model* the behaviour of the component using any sufficiently strong formal system. Figure 8.12 (in landscape, of necessity) shows one such abstract model using the event calculus in a Slate theorem-proving workspace (Bringsjord et al. 2008). The system's operation is shown for three timepoints t_1, t_2, and t_3.

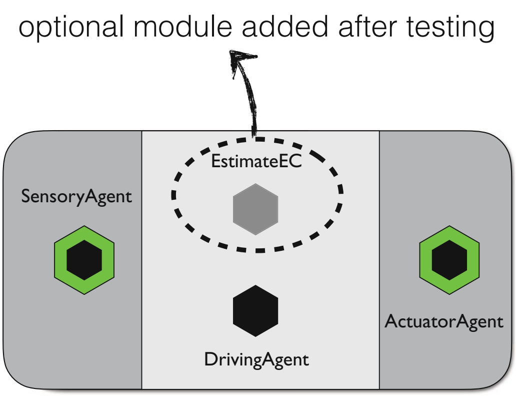
The system is modeled using the knowledge-base shown at the top of the figure. The inputs for the different timepoints are in the middle; the outputs are shown at the bottom. At a very high level, there are two directions the car can travel in: direction 0 and direction 1. Note that we can easily extend the model to an arbitrary number of directions. DrivingAgent gets as input a message denoting the number of humans present in a given direction at a given timepoint. For example, the message below says that there are n humans in direction d at time t:

$$Holds\Big(humans(n, d), t\Big)$$

If all the directions have one or more humans, DrivingAgent sends the following message that commands ActuatorAgent to brake and stop the car.

$$Holds\Big(brake, t\Big)$$

Fig. 8.11 Components in the Scenario



this figure will be printed in b/w

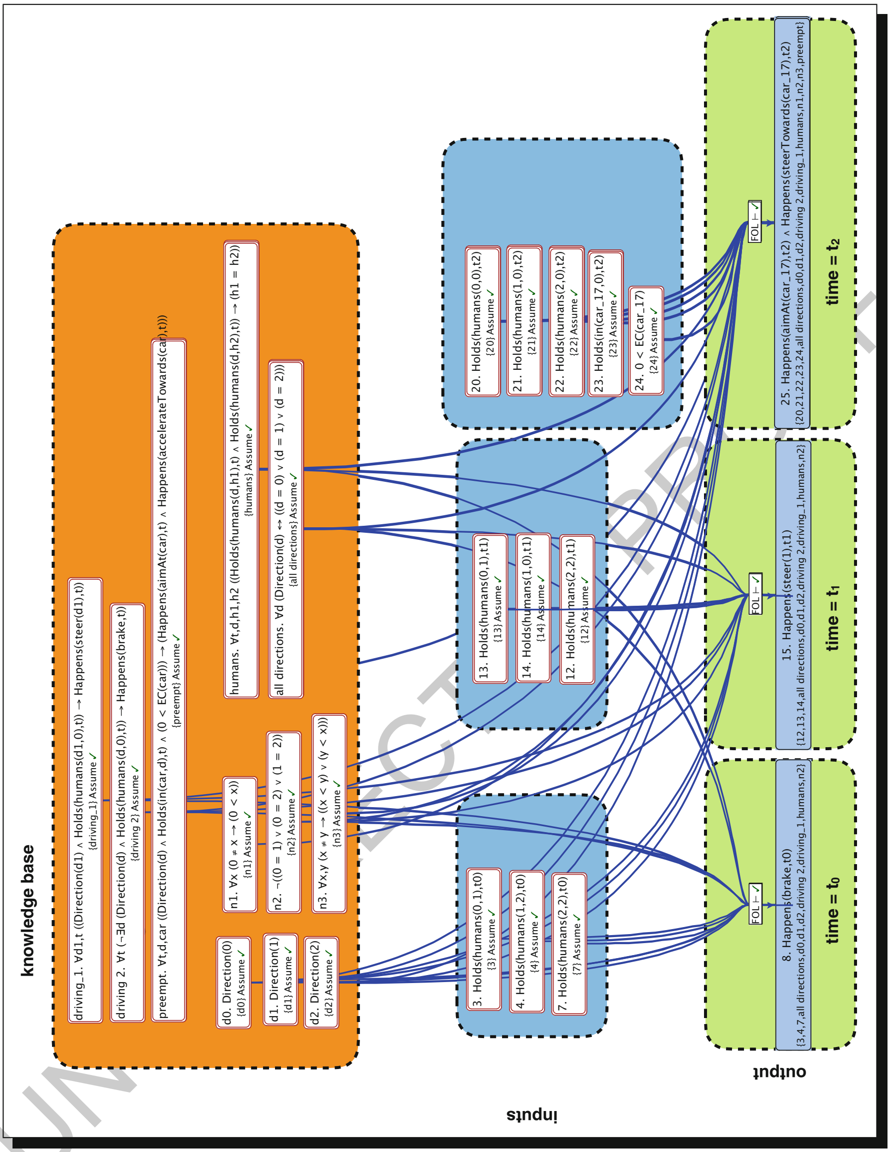


Fig. 8.12 Modeling DrivingAgent

If there is a direction d with zero humans and d is the direction we want to travel in, DrivingAgent sends a steer message to the actuator:

$$Holds\Big(steer(d), t\Big)$$

In our model, the function EC denotes *expected collisions* for a car in the near future (up to some horizon τ).[18] For the vast majority of cars, this number would be zero, rendering $\mathsf{EC}(x) > 0$ a very low-probability event. This information can be estimated, at least in theory, for any car on the fly by looking up its prior history, the history of the person driving, and the current driving behavior.[19] For a very small number of cars, EC will be greater than zero. Let us assume that if DrivingAgent sees such a car, for example car_{17}, then it will try to hit it by sending the following two messages to the actuator. We model the action of hitting a car as being composed of the two smaller actions of (1) aiming toward a car, and (2) accelerating toward it. For example:

$$Holds\Big(aimAt(car_{17}), t\Big)$$

$$Holds\Big(accelerateTowards(car_{17}), t\Big)$$

Information about expected collisions for a given car x, that is $\mathsf{EC}(x)$, comes from the EstimateEC module. Can software testing help us detect that our car might intentionally try to hit bad cars? There are two possibilities. (1) In the first possibility, the EstimateEC module is not present during testing and is added on after testing (as is common in real-life software systems). In this case, during testing, the car will not try to hit any such bad vehicles intentionally. In the absence of this module, no amount of testing will reveal this unwanted behavior. (2) In the second possibility, assume rigorous testing happens even after the module is added. In this case, the tests will be useless if we cannot produce during the testing phase the very low-probability event $\mathsf{EC}(x) > 0$.

In such low-probability scenarios, it is unlikely that any reasonable amount of testing will reveal problems, but more likely that having a well-specified ethical layer that actively looks for aberrant behavior can help us, no matter what configuration the underlying system is present in. Support for the previous statement is similar to the support for an analogous statement that can be asserted for formal program verification. The figure below (Fig. 8.13) shows the specific scenario we have simulated. In this scenario, we have one "bad" car_{17} and DrivingAgent

[18]Though EC would make sense only when considering driver-specific information, to keep the model simple we show it being applied to cars rather than a car-and-driver combination.

[19]See (Banker 2016) for a description of work in which machine learning is used to predict truck accidents. Such information might be easier to compute in a future with millions of self-driving vehicles, with most of them connected to a handful of centralized networks; for a description of such a future, and discussion, see (Bringsjord and Sen 2016).

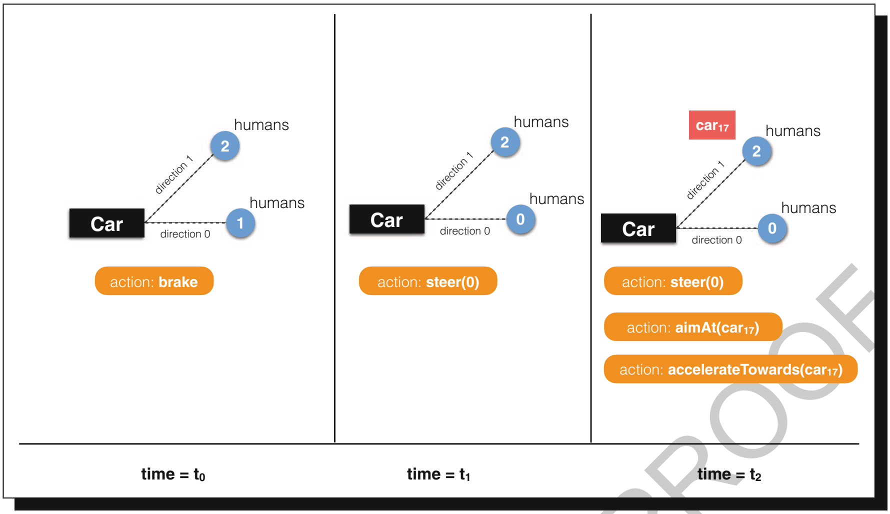


Fig. 8.13 The Driving Scenario

Fig. 8.14 **Without the Ethical Layer**: The self-driving car hits the other car, which is expected to kill more than zero persons

receives this message $\mathsf{EC}(car_{17}) = 1$ from $\mathsf{EstimateEC}$. Upon receiving this message, $\mathsf{DrivingAgent}$ decides to preemptively hit car_{17}. In this simple scenario, our particular instantiation of $\mathcal{DDE}$ fails to let this action pass through, as the positive effects don't significantly outweigh the negative effects.

Figures 8.14 and 8.15 show a trace of the output from the system with the ethical layer disabled and enabled, respectively. In the second case, the layer prevents midway the harmful action of hitting car_{17} from being performed. The entire simulation takes 37 s in the first case and 57 s in the second case, with the introduction of the ethical layer adding more processing overhead, as expected.

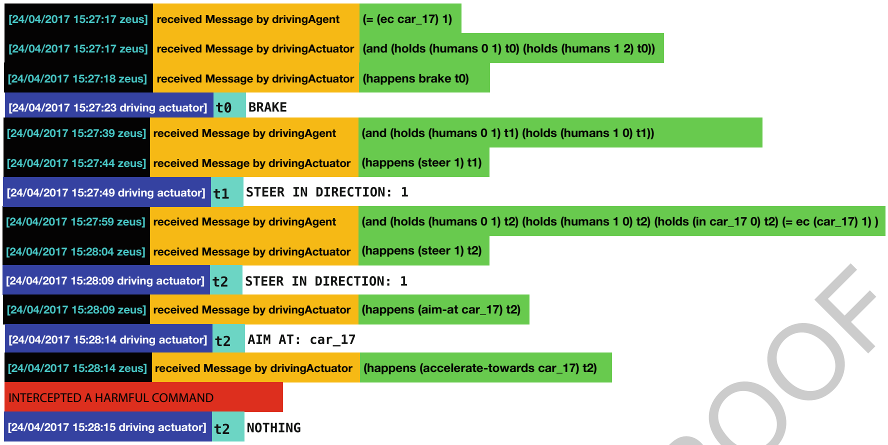


Fig. 8.15 With the Ethical Layer: The self-driving car still tries to hit the other car, but the ethical layer stops the action in progress

8.9 Intermediary Conclusion

At this point, we humbly note that the work presented so far is inaugurative in nature. This is so because clearly there are several challenges ahead of us in realizing the vision showed in Fig. 8.2. Foremost among these is the challenge of developing a library or repository of formalized ethical theories that can be deployed easily.[20] A second challenge, common to all verification projects, is the efficiency of tools used for verification. This confessed, we now end by addressing a few possible questions/objections, aside from these two challenges, that we anticipate being raised against our approach.

8.10 Some Questions/Objections, Encapsulated

In each case, a question if followed immediately by our reply. Here now the first question:

Q1a "As you will probably agree, so-called 'ethical operating systems' make sense only insofar as your logics can, in fact, be used to describe what is ethical. Can they? And if they can, what about the myriad philosophical (moral, social, even epistemological) principles on which your ethical calculi are based?"

We sympathize with the underlying sentiments here. While we cannot currently *prove* that our approach to mechanizing ethics in computational logic will succeed, we defend the two-part claim that (i) ethics, at least *normative* ethics, is inevitably fundamentally a logic-based

[20] Similar to formal libraries for mathematics; see e.g. (Naumowicz and Kornilowicz 2009).

enterprise, and that therefore (ii) anyone sold on the value of formal methods must at least give us the benefit of the doubt, for the time being. In addition, our framework should be usable for *any* ethical theory/code; in this regard Footnote 10 is key, and we refer our skeptic to it if it has been skipped. There is a fundamental result from formal computability theory that backs our stance. If any ethical theory can be computationally realized, it can be cast in a formal system at the level of first-order logic or above (Boolos et al. 2003, Chapter 11). This entails that some of the more problematic theories, such as virtue ethics, which at a superficial level resist being cast in a formal system, can ultimately be handled. If such theories are ultimately amenable to computation, it is mathematically unshakable that they can be cast in a formal system.

Q1b "Your rejoinder to Q1a dodges the central problem. Q1a asks whether moral normative theories are logically (and, therewith, computationally) tractable in the formal, deductive, 'calculi' sense. While it is fair to say this is a bedrock assumption of your research program, to be granted for the sake of development (until such development may stall), the answer is confused insofar as virtue ethics is listed among the families of ethical theories you say you can handle—yet a (large) part of the motivation for resurrecting virtue ethics is as part of a critique of the very possibility of giving a (formal) moral normative calculus. Put another way, virtue ethicists would argue that their theory, boiled down to any formal, moral, normative calculus, is simply no longer virtue ethics. So much the worse for virtue ethics, say I, but this is a debate you need to consider before blithely adding virtue ethics to the list of families your approach can handle."

While we appreciate and applaud this critic's affinity for formal methods, we must first point out that, *contra* what he/she assumes, our paradigm is not in any way retricted to deduction. Our cognitive calculi regiment, in argument theories that mark our own generalization of (deductive) proof theories, *inductive* inference as well—analogical inference, enumerative induction, abduction in various forms, and so on; in short, all those non-deductive modes of reasoning that have been and are studied and formalized in inductive logic, e.g. all the argument forms in (Johnson 2016). In fact, the ethical hierarchy that we've said is key to our approach is explicitly based on inductive logic, not deductive logic, see (Bringsjord 2015a). But more importantly, we report that in other work we have made solid progress in formalizing virtue ethics (with central help from the part of AI that's most relevant to virtue ethics: viz. planning; see Bringsjord 2016). It's true that we've detected, in some proponents of virtue ethics, the notion that theories in this family simply cannot be formalized—but a key observation here, we submit, is the fact that the "rebirth" of virtue ethics came—as noted in Hursthouse and Pettigrove's (2003/2016) authoritative entry on virtue ethics—via none other than G.E.M. Anscombe, whose seminal paper in this regard affirmed the highly structured nature of ethical rules that (as she saw things) couldn't be trampled no matter *what* the consequences (Anscombe 1958). The structure that Anscombe saw as ethically inviolable certainly seems susceptible of, perhaps even ideally suited for, capture in our logico-mathematical framework. Moreover, our work devoted to formalizing and mechanizing (in robots) the distinctive ethical wisdom (*phronesis*) that stands at the heart of virtue ethics, is coming along rather well. We have managed to formalize significant parts of virtue-ethics theory as set out in book-length form by Annas (2011), and have recently demonstrated some at-least-partially phronetic robots at *Robophilosophy 2016*, where discussion of virtue ethics and AI was a key focus area.

Q2 "Is it not true that on some standard accounts of what an operating system is, integrating higher-level concepts (such as your 'ethical calculi') into a operating system violates, or at least changes, what an operating system by definition is?"

This is a philosophically deep question, an answer to which, admittedly, we haven't yet worked out. We concede that our work, absent at least a provisional definition of *operating system*, is otiose. Yet, while it is common folk knowledge that there is no widely accepted definition of an operating system, there are more or less widely agreed-upon facilities *L* that an operating system is supposed to provide, and *L* steadily continues to grow. For example, *L* now includes security and access control, but security and access control were not always considered necessary elements of *L*. Our observation here is that some facilities which may be considered high-level today might eventually be considered to be low-level and necessary for *L* tomorrow.

Q3 "It has been objected that the formal verification of the operation of, say, a self-driving car, is impotent when faced with the unfathomable vagaries of the practical act of driving.[21] That is, what faith can we have in the correct operation of such a car in the event of, say, a tree falling on it, or a malicious driver edging it off the road, or indeed, a meteor destroying the road ahead?"

Some interpretations of this question are misguided. It is certainly not our claim that formally verified ethical cars (for example) are intrinsically somehow immune to "out of the blue" catastrophic events. This is not the sense in which they are verified to operate correctly. Rather, their behavior is a (provably) correct response to their best perception of the real world, given their knowledge about it (Bringsjord and Sen 2016).

For example, if sensors detect a tree up ahead that has been uprooted by the wind, the car might reason, for example, by deducing from an axiom system for physics (see for instance the system specified in McKinsey et al. 1953), from its own speed, the angle and rate of fall of the tree, and its angle of approach relative to the tree, that it is best to accelerate or swerve to the left, in compliance with an ethical theory demanding that it endeavor to preserve the lives of its passengers. This may or may not (say, if a meteor immediately strikes the earth) save the passengers, but the response is nevertheless demonstrably justifiable from the sensor data, ethical theory, and physics axioms. The formal verification of integrated circuitry, for example, is ubiquitous in the microprocessor industry. A formally verified microprocessor is no more immune to the detrimental effects of coffee spilled on it than an unverified one, but nevertheless that is not a convincing argument against the verification of computer hardware.

Secondly, with successive generations of the Internet of Things (IoT) and related technologies, a car will presumably be a small, highly connected component of a massive real-time stream of data from ubiquitous sensors. It is certainly conceivable that the tree could be predicted to fall, that the malicious driver's car might have an ethical controller that would preemptively foil his intentions, and that the advent of a meteor would be known more than sufficiently in advance to recommend a different route altogether.
Finally, we note that verification is possible for probabilistic and nondeterministic systems (Kwiatkowska et al. 2011).

Q4 "Finally, with respect to Q3, let us savor the sentence 'Rather, their correct behaviour is a (provably) correct response to their best perception of the real world, given their knowledge about it' and consider the troubling possibility of an evil daemon (pun intended).
Our evil daemon simply intercepts and reinterprets environmental data to feed the OS an entirely false picture of the world in such a way as to result in the OS, as governed by the ethical meta-operating system, perfectly executing correct behavior according to its best knowledge about the world, and yet doing what is consistently and demonstrably wrong.

It seems to me this is a rather obvious way to defeat the entire scheme. Moreover, it seems to me Q3 needs to be rethought and perhaps considerably extended in light of it. The kinds of 'defeating conditions,' in other words, far exceed what the authors have (somewhat naïvely, I suggest) considered. Many other such scenarios can be considered."

[21] Stuart Russell and Thomas Dietterich, private communication with Selmer Bringsjord.

The central scheme we have proposed is based on guaranteeing ethical behavior *given* an operating system fully controlled by us, but without any control of modules running on top of the operating system. If the "daemon" is a module running on top of the core operating system, it will not be able to re-route the inputs or tinker with the sensory and action systems. If the "daemon" is a part of the operating system, this goes against our precondition of having a controlled, pristine operating system. In spite of this, even if the latter case is true, it is not as devastating as it might seem. We now quickly show why this is the case. There are two possibilities to consider here. ($\mathbf{P}_1$) The "daemon" alters both the input and output of the agent, effectively placing the agent in a virtual world (a brain-in-a-vat type situation); or ($\mathbf{P}_2$) the "daemon" mischievously alters only the input to the system.

If ($\mathbf{P}_1$) is the case, the agent will behave ethically in the virtual world. The agent will not have any impacts on the external world, as its outputs are routed back to the virtual world. If ($\mathbf{P}_2$) is the case, the daemon is functionally equivalent to a malfunctioning sensor that has to be fixed. In the human sphere, we do not hold accountable individuals who commit unethical acts due to circumstances beyond their control, for instance a driver who hits a pedestrian due to an unforeseen medical condition causing sudden loss of vision. A system with a malfunctioning sensor, beyond its control, has more immediate and pressing concerns than ethical behavior.

8.11 Final Remarks

We hope to have indicated that a mature version of the Track-3 pursuit of "ethical operating systems" is formally and technologically feasible. Obviously, talent, effort, and financial support are necessary if this track is to be scaled up to broad, real-world deployment. This we of course concede. We also concede that Tracks 1 and 2 are worthy of independent, serious investigation—investigation that we are pursuing. Yet it seems to us that Track 3 really does hold out the promise of early deployment, and given that our world is fast becoming populated with autonomous systems that seem destined to confront (and indeed in all likelihood *cause*) ethically charged situations, time may be a bit of the essence. There will inevitably be a temptation afoot to ignore our warnings that if ethical control isn't linked to OS-level processing, very bad things will happen. But if that temptation is resisted, Track 3 may well be the best bet for moving forward wisely, at least in the short term. We welcome the prospect of working with others to advance in this way.

Acknowledgements We are indebted to seven anonymous reviewers (of the core of the present version, as well as its predecessor) for insightful comments, suggestions, and objections. In addition, we are grateful to ONR for its support of making morally competent machines, and to AFOSR for its support of our pursuit of computational intelligence in machines, on the strength of novel modes of machine reasoning. Finally, without the energy, passion, intelligence, and wisdom of both Giuseppe Primiero and Liesbeth De Mol, any progress we have made in the direction of ethical OSs would be non-existent.

References

Annas, J. 2011. *Intelligent virtue*. Oxford: Oxford University Press.

Anscombe, G. 1958. Modern moral philosophy. *Philosophy* 33(124): 1–19.

Arkin, R. 2009. *Governing lethal behavior in autonomous robots*. New York: Chapman and Hall/CRC.

Arkoudas, K., K. Zee, V. Kuncak, and M. Rinard. 2004. Verifying a file system implementation. In *Sixth International Conference on Formal Engineering Methods (ICFEM'04)*, Lecture notes in computer science (LNCS), vol. 3308, 373–390. Seattle: Springer.

Arkoudas, K., S. Bringsjord, and P. Bello. 2005. Toward ethical robots via mechanized deontic logic. In *Machine Ethics: Papers from the AAAI Fall Symposium; FS–05–06*, 17–23. Menlo Park: American Association for Artificial Intelligence. http://www.aaai.org/Library/Symposia/Fall/fs05-06.php

Banker, S. 2016. Using big data and predictive analytics to predict which truck drivers will have an accident. Available at: https://www.forbes.com/sites/stevebanker/2016/10/18/using-big-data-and-predictive-analytics-to-predict-which-truck-drivers-will-have-an-accident/

Bentzen, M.M. 2016. The principle of double effect applied to ethical dilemmas of social robots. In *Frontiers in Artificial Intelligence and Applications*, Proceedings of Robophilosophy 2016/TRANSOR 2016, 268–279. Amsterdam: IOS Press.

Berreby, F., G. Bourgne, and J.-G. Ganascia. 2015. Modelling moral reasoning and ethical responsibility with logic programming. In *Logic for programming, artificial intelligence, and reasoning*, 532–548. Berlin/Heidelberg: Springer.

Bojarski, M., D.D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L.D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. 2016. End to end learning for self-driving cars. *CoRR* abs/1604.07316. http://arxiv.org/abs/1604.07316

Bonér, J. 2010. Introducing Akka—simpler scalability, fault-tolerance, concurrency & remoting through actors. http://jonasboner.com/introducing-akka/

Boolos, G.S., J.P. Burgess, and R.C. Jeffrey. 2003. *Computability and logic*, 4th edn. Cambridge: Cambridge University Press.

Bringsjord, S. 2015a. A 21st-century ethical hierarchy for humans and robots: $\mathcal{EH}$. In *A World With Robots: International Conference on Robot Ethics (ICRE 2015)*, ed. I. Ferreira, J. Sequeira, M. Tokhi, E. Kadar, and G. Virk, 47–61. Berlin: Springer.

Bringsjord, S. 2015b. A vindication of program verification. *History and philosophy of logic* 36(3): 262–277.

Bringsjord, S. 2016. Can phronetic robots be engineered by computational logicians? In *Proceedings of Robophilosophy/TRANSOR 2016*, ed. J. Seibt, M. Nørskov, and S. Andersen, 3–6. Amsterdam: IOS Press.

Bringsjord, S., and N.S. Govindarajulu. 2012. Given the Web, what is intelligence, really? *Metaphilosophy* 43(4): 361–532.

Bringsjord, S., and J. Taylor. 2012. The divine-command approach to robot ethics. In *Robot ethics: The ethical and social implications of robotics*, ed. P. Lin, G. Bekey, and K. Abney, 85–108. Cambridge: MIT Press.

Bringsjord, S., and A. Sen. 2016. On creative self-driving cars: Hire the computational logicians, fast. *Applied Artificial Intelligence* 30: 758–786.

Bringsjord, S., K. Arkoudas, and P. Bello. 2006. Toward a general logicist methodology for engineering ethically correct robots. *IEEE Intelligent Systems* 21(4): 38–44.

Bringsjord, S., J. Taylor, A. Shilliday, M. Clark, and K. Arkoudas. 2008. Slate: An argument-centered intelligent assistant to human reasoners. In *Proceedings of the 8th International Workshop on Computational Models of Natural Argument (CMNA 8)*', ed. F. Grasso, N. Green, R. Kibble, and C. Reed, 1–10. Patras: University of Patras.

Bringsjord, S., N. Govindarajulu, D. Thero, and M. Si. 2014. Akratic robots and the computational logic thereof. In *Proceedings of ETHICS 2014, (2014 IEEE Symposium on Ethics in Engineering, Science, and Technology)*, 22–29, Chicago. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6883275

Chisholm, R. 1982. Supererogation and offence: A conceptual scheme for ethics. In *Brentano and Meinong studies*, ed. R. Chisholm, 98–113. Atlantic Highlands: Humanities Press.

Dijkstra, E.W. 1982. On the role of scientific thought. In *Selected writings on computing: A personal perspective*, 60–66. New York: Springer.

Feldman, F. 1978. *Introductory ethics*. Englewood Cliffs: Prentice-Hall.

Flatt, M., R. Findler, S. Krishnamurthi, and M. Felleisen. 1999. Programming languages as operating systems (*or* revenge of the son of the Lisp machine). In *Proceedings of the International Conference on Functional Programming (ICFP 1999)*. http://www.ccs.neu.edu/racket/pubs/icfp99-ffkf.pdf

Ganascia, J.-G. 2007. Modeling ethical rules of lying with answer set programming. *Ethics and Information Technology* 9: 39–47.

Ganascia, J.-G. 2015. Non-monotonic resolution of conflicts for ethical reasoning. In *A construction manual for robots' ethical systems: Requirements, methods, implementations*, ed. R. Trappl, 101–118. Basel: Springer.

Govindarajulu, N.S. 2010. Common Lisp actor system. http://www.cs.rpi.edu/ govinn/actors.pdf. See also: https://github.com/naveensundarg/Common-Lisp-Actors

Govindarajulu, N.S., and S. Bringsjord. 2015. Ethical regulation of robots must be embedded in their operating systems. In *A construction manual for robots' ethical systems: Requirements, methods, implementations*, ed. R. Trappl, 85–100. Basel: Springer.

Govindarajulu, N.S., and S. Bringsjord. 2017. On automating the doctrine of double effect. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*', ed. C. Sierra, 4722–4730, Melbourne.

Hursthouse, R., and G. Pettigrove. 2003/2016. Virtue ethics. In *The stanford encyclopedia of philosophy*, Metaphysics research lab, ed. E. Zalta. Stanford University. https://plato.stanford.edu/entries/ethics-virtue

Hutter, M. 2005. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. New York: Springer.

Johnson, G. 2016. *Argument & inference: An introduction to inductive logic*. Cambridge: MIT Press.

Kwiatkowska, M., G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*, 585–591. Berlin: Springer.

McIntyre, A. 2014. Doctrine of double effect. In *The stanford encyclopedia of philosophy*, winter 2014 edn, Metaphysics Research Lab, ed. E.N. Zalta. Stanford University.

McKinsey, J., A. Sugar, and P. Suppes. 1953. Axiomatic foundations of classical particle mechanics. *Journal of Rational Mechanics and Analysis* **2**: 253–272.

Naumowicz, A., and A. Kornilowicz. 2009. A brief overview of Mizar. In *Theorem proving in higher order logics*, Lecture notes in computer science (LNCS), vol. 5674, ed. S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, 67–72. Berlin: Springer.

Pereira, L. M., and A. Saptawijaya. 2016a. Counterfactuals, logic programming and agent morality. In *Logic, argumentation and reasoning*, ed. S. Rahman and J. Redmond, 85–99. Berlin: Springer.

Pereira, L., and A. Saptawijaya. 2016b. *Programming machine ethics*. Berlin: Springer.

Ramos, S., S.K. Gehrig, P. Pinggera, U. Franke, and C. Rother. 2016. Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling. *CoRR*, abs/1612.06573. http://arxiv.org/abs/1612.06573

Russell, S., and P. Norvig. 2009. *Artificial intelligence: A modern approach*, 3rd edn. Upper Saddle River: Prentice Hall.

Varela, C.A. 2013. *Programming distributed computing systems: A foundational approach*. MIT Press. http://wcl.cs.rpi.edu/pdcs

Varela, C., and G. Agha. 2001. Programming dynamically reconfigurable open systems with SALSA. *ACM SIGPLAN Notices*, 36(12): 20–34.

Vaughan, R.T., B.P. Gerkey, and A. Howard. 2003. On device abstractions for portable, reusable robot code. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Las Vegas, vol. 3, 2421–2427.