From Coder to Creator: Responsibility Issues in Intelligent Artifact Design

Andreas Matthias

University of Kassel, Germany

Institute of Philosophy & Computing Centre

Abstract

Creation of autonomously acting, learning artifacts has reached a point where humans cannot any more be justly held responsible for the actions of certain types of machines. Such machines learn during operation, thus continuously changing their original behaviour in uncontrollable (by the initial manufacturer) ways. They act without effective supervision and have an epistemic advantage over humans, in that their extended sensory apparatus, their superior processing speed and perfect memory render it impossible for humans to supervise the machine's decisions in real-time. We survey the techniques of artificial intelligence engineering, showing that there has been a shift in the role of the programmer of such machines from a coder (who has complete control over the program in the machine) to a mere creator of software organisms which evolve and develop by themselves. We then discuss the problem of responsibility ascription to such machines, trying to avoid the metaphysical pitfalls of the mind-body problem. We propose five criteria for purely legal responsibility, which are in accordance both with the findings of contemporary analytic philosophy and with legal practise. We suggest that Stahl's (2006) concept of "quasi-responsibility" might also be a way to handle the responsibility gap.

Introduction

Since the dawn of civilization, man has lived together with artifacts: tools and machines he himself has called into existence. These artifacts he has used to extend the range and the quality of his senses, to increase or replace the power of his muscles, to store and transmit information to others, his contemporaries or those yet to be born. In all these cases, he himself had been the controlling force behind the artifacts' actions. He had been the one to wield the hammer, to handle the knife, to look through the microscope, to drive a car, to flip a switch to turn the radio on or off. Responsibility ascription for whatever the machines "did" was straightforward, because the machines could not act by themselves. It was not the machine which acted, it was the controlling human. This not only applied to the simple tools, like hammers and knives, but also to cars and airplanes, remotely controlled planetary exploration vehicles and, until recently, computers.

Any useful, traditional artifact can be seen as a finite state machine: its manufacturer can describe its range of expected actions as a set of transformations that occur as a reaction of the artifact to changes in its environment ("inputs"). The complete set of expected transformations is what comprises the *operating manual* of the machine. By documenting the reactions of the machine to various valid input patterns, the manufacturer renders the reader of the operating manual capable of effectively *controlling* the device. This transfer of control is usually seen as the legal and moral basis of the transfer of *responsibility* for the results of the machine's operation from the manufacturer to the operator (Fischer & Ravizza, 1998). If the operation of a machine causes damage, we will ascribe the responsibility for it according to who was in control of the machine at that point. If the machine operated correctly and predictably (that is, as documented in the operating manual), then we will deem its operator responsible. But if the operator can show that the machine underwent a significant transformation in its state which was not documented in the operating manual (e.g. by exploding, or failing to stop when brakes were applied) then we would not hold the operator responsible any longer, and precisely for the reason that he did not have sufficient *control* over the device's behaviour to be able to assume full responsibility for the consequences of its operation.

With the advent of *learning, autonomously acting* machines, all this has changed more radically than it appears at first sight. Learning automata, as we will see, are not just another kind of machine, just another step in the evolution of artifacts from the spear to the automobile. Insofar as responsibility ascription is concerned, learning automata can be shown to be machines *sui generis,* in that the set of expected transformations they may undergo during operation cannot be determined in advance, which translates to the statement that the human operator cannot *in principle* have sufficient control over the machine to be rightly held responsible for the consequences of its operation.

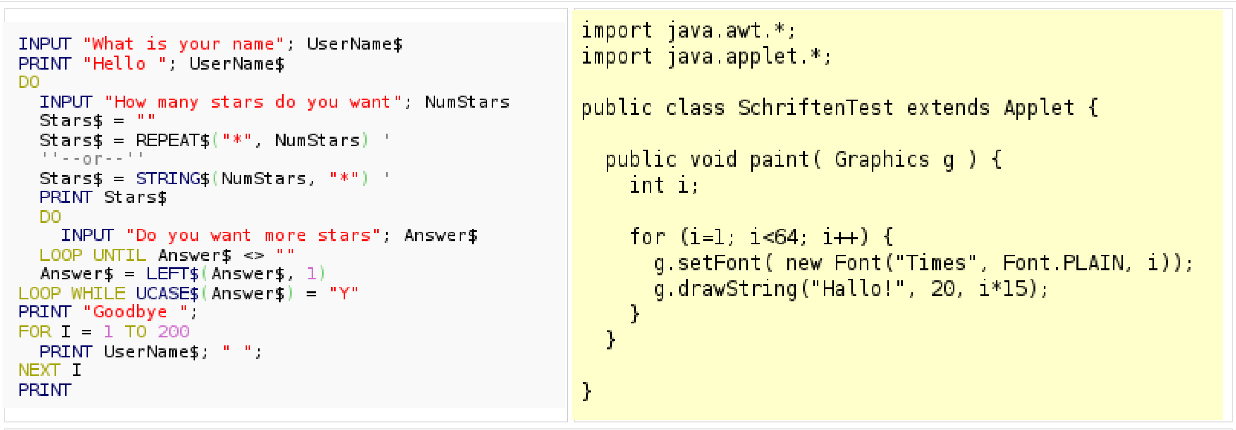
Learning automata cause a *paradigm shift* in the creation, operation and evaluation of artifacts. In the progress of programming techniques from classic, imperative programming, to declarative languages, artificial neural networks, genetic algorithms and autonomous agent architectures, the manufacturer/programmer step by step gives up control over the machine's future behaviour, until he finds her role reduced to that of a *creator* of an autonomous organism rather than the powerful, controlling *coder* that she still is in popular imagination and (all too often) in unqualified moral debate.

In the course of this chapter, we will retrace the crucial points of this technological development. We will see how exactly the shift from coder to creator takes place and what this means for the problem of responsibility ascription for the actions of learning automata. It can be shown that the loss of control over the operation of such machines creates a "responsibility gap" which must somehow be bridged. Since humans cannot have enough control over the machine's behaviour to rightly assume responsibility for it, we will examine the question whether learning, autonomous machines could possibly be ascribed themselves responsibility for their own actions. We will discuss the prerequisites to machine responsibility and see that it does not necessarily mean that we will need to consider machines to be moral agents or even quasi-personal entities. Instead, responsibility ascription to a machine can be done without a shift in the metaphysical status of the machine using a "functional" approach to responsibility ("quasi-responsibility," as put forth by Stahl, 2006): if we understand the concept of responsibility to be a social construct which serves particular social aims, then it seems feasible to apply it to machines, as long as those aims are served and the corresponding social goals fulfilled.

Background

Learning machines

We are surrounded by machines which act on our behalf, more or less autonomously, and quite often we are not even aware of it. All around us are things like electronic noses for banana ripeness determination (Llobet, 1999), collision avoidance systems in automatic submarine navigation (Schultz, 1991a), autonomously flying (Stancliff & Nechyba, 2000), game playing (De Jong, 1988), and web document filtering (Zhang & Seo, 2001) machines and programs, while for a long time elevators in high-rise office towers are being controlled by adaptive, learning programs, which are able to predict and anticipate traffic patterns and to modify their own behaviour to better serve those patterns (Sasaki et al., 1996).



```
INPUT "What is your name"; UserName$
PRINT "Hello "; UserName$
DO
  INPUT "How many stars do you want"; NumStars
  Stars$ = ""
  Stars$ = REPEAT$("*", NumStars) '
  ''--or--''
  Stars$ = STRING$(NumStars, "*") '
  PRINT Stars$
  DO
    INPUT "Do you want more stars"; Answer$
  LOOP UNTIL Answer$ <> ""
  Answer$ = LEFT$(Answer$, 1)
LOOP WHILE UCASE$(Answer$) = "Y"
PRINT "Goodbye ";
FOR I = 1 TO 200
  PRINT UserName$; " ";
NEXT I
PRINT
```

```
import java.awt.*;
import java.applet.*;

public class SchriftenTest extends Applet {

  public void paint( Graphics g ) {
    int i;

    for (i=1; i<64; i++) {
      g.setFont( new Font("Times", Font.PLAIN, i));
      g.drawString("Hallo!", 20, i*15);
    }
  }

}
```

Figure 1: Imperative programming in BASIC and Java. It can easily be seen that the programs consist of commands to the machine (INPUT, PRINT , setFont) which are executed from top to bottom or in loops (DO LOOP UNTIL). This is the common structure of all imperative programming, regardless of the language used. (BASIC code from Wikipedia)

In order to understand why some of these machines cannot be dealt with using traditional patterns of responsibility ascription, it will be necessary to have a look at how these machines are being constructed and programmed. Unfortunately, there is still widespread confusion regarding this point, even in the philosophical literature discussing the status of machines and the ethics of man-machine interaction. Only by understanding the broad principles of the construction of learning automata, we will be able to see where they differ from traditional artifacts and why they are capable of inducing a paradigm shift in the role of the engineer as well as in the question of responsibility ascription for the consequences of their operation.

In the present context, we can distinguish five main categories of computer software: traditional, procedural programs, declarative programming, connectionist architectures (including reinforcement learning systems), genetic algorithms (including genetic programming), and autonomous agents (mobile and immobile). As we will see, each one of these technologies represents one further step in the process of manufacturer and operator control loss.

Procedural programming

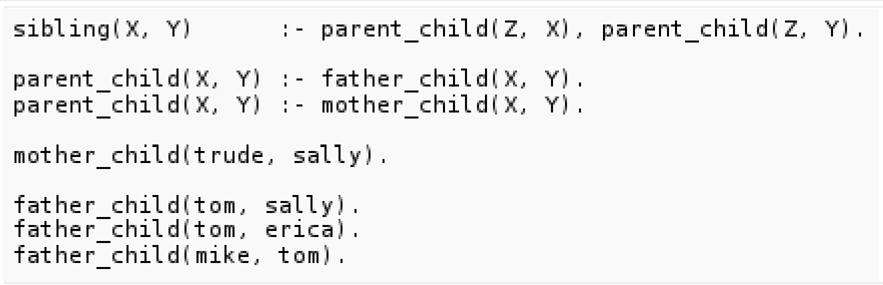
In this "classic" model of software development, the programmer writes down her program as a series of commands to the machine. Each command triggers a corresponding, well-defined action of the computer system: the transfer of data from one memory location to another, arithmetic operations, comparisons of the contents of variables, input or output of data from or to the operator of the program. Programs written in Basic, C, Cobol, Fortran, Pascal or Java all are built according to this principle (see figure 1). The reader of such a program can look at the code, "execute" it mentally step by step, and thus anticipate what the machine is supposed to do. The machine, too, does not do anything mysterious or incomprehensible: it just executes the commands one after the other in a well-defined order. An analogy for the process of programming in an imperative language would be to cook following a detailed recipe. The cook does not do anything except executing, step by step, the instructions in the recipe, until he reaches the end of it, which should coincide with the completion of the meal that was described therein.

In this programming model, the programmer still has the complete control over the machine's behaviour, and every action of the *program* is really an action of the *programmer,* as the program itself cannot deviate from the prescribed sequence of operations.

Declarative programming

Declarative programming was widely introduced in the 1970's. It aims to change the role of the programmer, in that she should not any more describe to the computer *how* to do things, but rather *what* the data are and what relations exist between them. A program in a declarative, logic-oriented programming language would look much like a series of statements in a predicate logic notation, which describes some part of the world. The interpreter of the programming language itself would then apply the facts and relations given in the program in order to find solutions to users' queries (see figure 2).

As an analogy we could imagine describing a family tree to a stranger, telling him how many sisters and brothers we have, their names and ages, and also the data of the parents and their siblings. After we have supplied the listener with this information, he is able to answer complex questions about the family tree (e.g. "how many sisters does Mary have, which are older than twenty-five?"), by using a process of inference from the known facts. He does this without the need for us to supply him with a step-by-step recipe for executing this process of inference, because he has "built-in" rules for doing logical operations of this kind.



```
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).
```

Figure 2: A declarative program in PROLOG. Here, the programmer does not issue commands to the machine. Instead, she states a series of facts, from which the machine will derive a program. (PROLOG code from Wikipedia)

A typical application of the declarative programming paradigm are *expert systems,* which consist of a knowledge base about a particular domain and an inference engine, which is capable of drawing logical conclusions from the facts stored in the knowledge base. Expert systems are used in areas where the domain knowledge can be easily extracted from a human expert and converted into logical rules: so for example in medical diagnosis, or in the diagnosis of motor faults. One of the earliest expert systems was DENDRAL, developed at Stanford in the late 1960's. DENDRAL was designed to infer the structure of organic molecules based on their chemical formulas and mass spectrographic information about the molecule. It proved so successful that similar systems are still used in chemical laboratories today. Another famous system, MYCIN, also built at Stanford, was aimed at diagnosing bacterial infections of the blood. MYCIN could provide explanations for its reasoning and handle uncertain input data (Luger, 2005).

For the present discussion it is only important to note that declarative programming and expert system knowledge bases deprive the programmer from one aspect of control over his program: the control over the flow of execution. Unlike with imperative languages, the programmer of a declarative program does not know exactly *how* it will be executed, in what exact order the clauses and predicates are going to be applied by the machine in order for it to "prove" the users' queries. Also, like with any extensive system of formal logic statements, the programmer cannot anticipate every possible production from the given axioms and rules of inference, just because of the sheer number of possible productions.

Connectionist systems

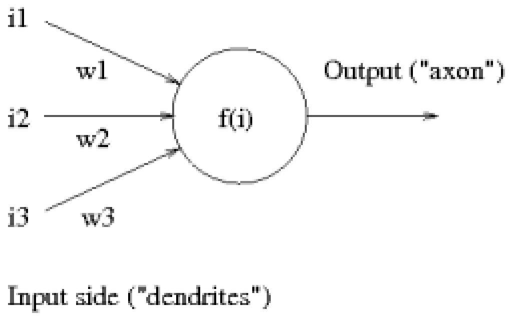


Figure 3: Basic structure of a single artificial neuron. The input values i1-i3 are multiplied by weights w1-w3 and the result is used as the parameter of a trigger function f(i). If the function "fires", a signal is output along the axon. This signal can, in turn, be used as an input for other artificial neurons.

Connectionist architectures are increasingly used in all domains where the representation of the domain knowledge in the form of clear and distinct symbols and formal rules is not possible. Artificial neural networks consist of hundreds or thousands of artificial "neurons", which are often designed as shown in figure 3: The neuron, which is simulated in software, has a number of input channels (in the biological metaphor corresponding to its dendrites). The input signals are multiplied with adaptable weights (w1-w3) and the sum of them is then used as a parameter to a function, which maps the sum of the weighed inputs to an output signal. The output signal is in turn used as input to other artificial neurons. Most often such artificial neurons are arranged into layers (see figure 4).

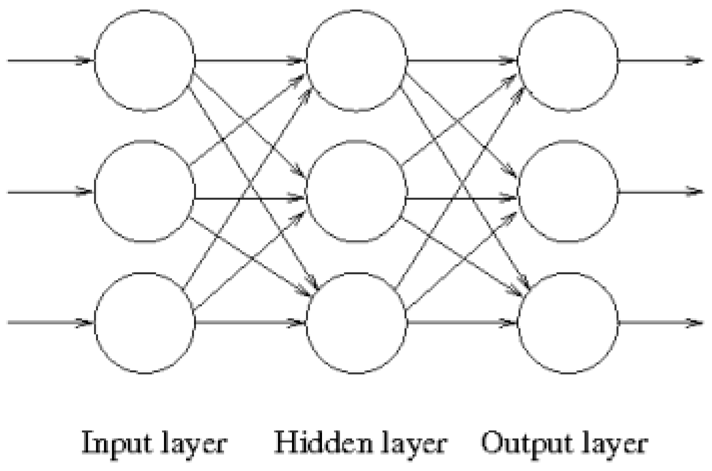


Figure 4: A three-layer artificial neural network.

By adapting the weights at its "synapses" the neural network can *learn.* It is important to see that neural networks don't contain a symbolic representation of knowledge anywhere inside them. In order to learn, they are presented with pairs of input/output patterns, and then they adapt the weights of their synapses so that the given input pattern leads to the desired output pattern. This is repeated for all pairs of input and output the system is supposed to learn. After training is completed, the artificial neural net is not only capable of mapping the learned input patterns to the desired output; it can also map previously unseen input patterns to the outputs of *similar* input patterns. This is an important feature: it enables the network to "recognize" and classify noisy and varying input correctly. Accordingly, artificial neural networks are mainly used for pattern recognition tasks: to recognize handwriting in palmtop computing devices, to identify faces of people in video images, to convert the scanned image of a page back to computer-readable text.

In order to "program" a neural network to do such a task, we just present it with a great number of examples, along with the result we wish it to output for every example ("learning phase"). That is, if we wanted a neural network to learn to recognize hand-written input, we would present it with a few hundred examples of hand-written letters from different writers, and the network would automatically adjust its internal weights so that it is able to match the supplied input patterns to the characters they are supposed to depict. After a few thousand training rounds for each pattern (which would not need more time than a few minutes to complete), the network would be able to recognize hand-written letters, although the programmer never told it *how* to achieve this result. The *how* is something the network figured out "by itself", by adjusting its internal weights so as to be able to match each input pattern to the desired output during the learning phase.

The important thing here is to note that the behaviour of artificial neural networks is not *programmed* any more. Instead, it is *taught by example.* Inside the network there is no symbolic representation of any learned facts and rules, but only an ordered set of connection weights, which somehow encode the network's performance. But those weights cannot be examined by a human in order for her to predict its future behaviour. They have no symbolic interpretation. In fact, the teacher of an artificial neural network can never fully predict how the network will process a future input. She can only train the network with as many different sets of inputs as possible, in the hope that she will cover the whole range of significant variations in the input patterns. But, and in this she is much like a teacher of humans, she cannot look *inside* the network to see the stored knowledge. As we do with children in school, the teacher of a neural network can only apply *tests* to the network and evaluate its answers to the test input patterns.

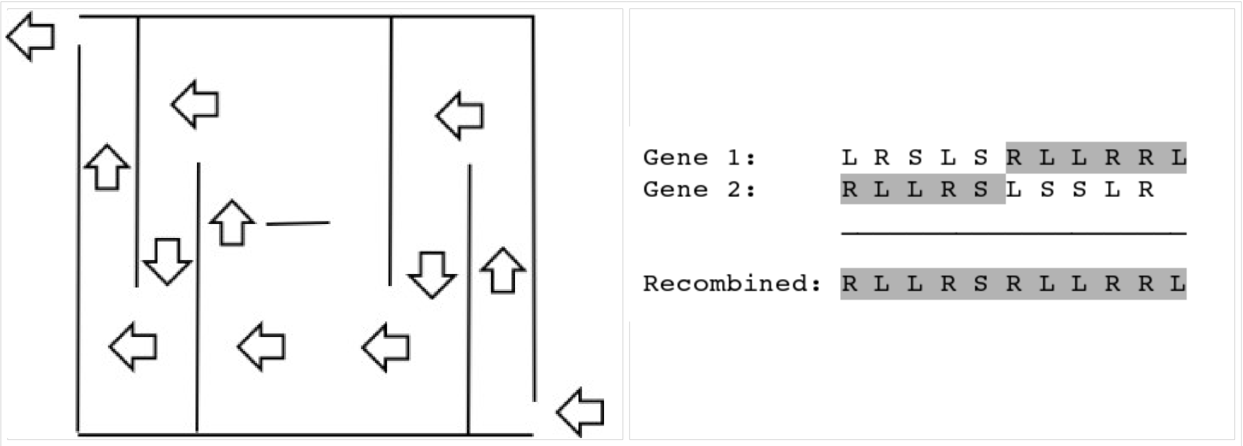
Another often used technology is *reinforcement learning* (Moriarty, 1999). Here the distinction between the phases of training and production use of the neural net is lifted. The network learns and adapts its reactions inside its final operating environment, by exploring available action alternatives in a trial-and-error fashion and optimising its own parameters according to the results. Thus, the exploration phase is an integral part of the design of the working machine and cannot be separated from it. This is necessary in highly dynamic environments, where the system has to adapt to continuous change in order to achieve optimal performance. Imagine, for example, a system which controls the flow of traffic on highways. Such a system needs not only to adapt to changing traffic patterns in the course of a day, a week, and a month, but also to the varying availability of the roads themselves: floods, accidents, public works are only some of the causes for change in the infrastructure that is available to the system to work with. A statically trained program would not be able to adapt to such changing conditions while maintaining a good performance. Reinforcement learning, on the other side, introduces *trial-and-error* into the operating phase. It continuously tries out new alternative actions and evaluates the resulting performance of the system it controls. Good results "reinforce" the system's exploratory reactions, bad results trigger further exploration of other alternative actions. While reinforcement learning systems are able to control a wide range of highly dynamic situations well, they *inevitably* introduce erroneous decisions into the process of control. "This is quite contrary to the common, traditional understanding that technology, done correctly, must operate free of errors, and that errors are always the errors of the programmer, not of the programmed machine. Reinforcement learning presents us, for the first time, with the *necessity* of errors as we know it from living systems: as a system *feature*, the precondition for learning and adaptive behaviour, and not merely a product flaw" (Matthias, 2004a).

Genetic algorithms

In contrast to artificial neural networks, which model software after the general principles of operation of the nervous system, genetic algorithms (Holland, 1975) mimic the process of evolution through variation, genetic recombination and selection. The solution to a problem is represented as a chain of symbols, that together comprise the "genome" of a virtual organism. If, for example, the problem is to find a way through a maze, a successful solution would be a chain of symbols that denote direction changes: right, right, left, right, left, and so on, until the maze has been successfully traversed (see figure 5). The solution is arrived at by generating a big initial population of virtual organisms, each one with its own, random genome of direction changes. These organisms are now put into the virtual maze, where some of them get immediately stuck at the walls, some take a few steps ahead, and some might even take a successful turn or two. The most successful organisms (the ones which take the greatest number of steps before they collide with a wall) "mate" with each other in such a way, that their "genetic information" is combined in their offspring, often by a simple cross-over of parental genes: the chains of direction statements of the parents is cut at some point, and the offspring receive parts of the genetic information of both parents. This process is repeated, often over hundreds or thousands of simulated generations, until some individuals contain in their genome a string of symbols which represents one solution to the problem. Since the generations can be simulated at high speed, such a genetic algorithm might even arrive at a solution *faster* than a conventional programmer, who would first need to solve the problem at hand himself, and then implement his algorithm in software. Genetic algorithms are robust: they can be run again and again when conditions change, and they will always evolve some solution to the problem at hand. Unfortunately, there is no guarantee that this would be an *optimal* solution. Like biological evolution, genetic algorithms can get stuck in relative optima: solutions that are relatively better than others, but not the globally best solutions.

Applications of genetic algorithms include the evolution of robot behaviours (Schultz, 1994), the classification of sensory input (Stolzmann et al., 2000), and the navigation of autonomous vehicles (Schultz, 1991b).

Genetic algorithms are, for the programmer, another step away from the role of the coder: she does never get to *see* the actual code that is going to be executed, because the code evolves by itself under the constraints defined by the environment. The programmer only sets up this environment and generates a big number of random individuals, which are then left to themselves, to breed and evolve as biological organisms would do. Here we have clearly reached the stage where the programmer is but the *creator* of a simulated world, and not a *coder* anymore. She is no longer in the position to rightly assume responsibility for the code that is to evolve inside the organisms she has created.



```
Gene 1:      L R S L S R L L R R L
Gene 2:      R L L R S L S S L R
             _____________________
Recombined:  R L L R S R L L R R L
```

Figure 5: Traversing a maze with a genetic algorithm. Left: the way through the maze. Right: two genomes recombine to produce an offspring which correctly represents the desired solution (a way through the maze). The individual genes denote directions: S: go straight, R: turn right, L: turn left.

Autonomous agents

Autonomous agents don't present any new features compared to the other technologies discussed, except that they are designed to act autonomously, *out of the reach and supervision* of the programmer. There are both embodied agents (like autonomous, mobile robots, which can physically move around in space and manipulate objects; see Adams, 2000) and pure information agents, which move in a virtual space created inside some computer infrastructure (e.g. Internet search engine crawlers).

"Two points concerning agents are interesting in the present context: First, that agents are *per definitionem* designed to act, and that, in the course of their operation, they must inevitably interact with other things, people, and social entities (laws, institutions, expectations). Second, that agents which have a physical presence constitute a new category of machines: such that can learn from the direct interaction with a real environment and that can in return directly manipulate this same environment. These machines have (contrary, for example, to desktop computers) an unmediated access to sensory impressions, their symbolic representation, and subsequent actions that lead to new sensory impressions, and they act in the same environment as humans do" (Matthias, 2004a).

The paradigm shift: from coder to creator

Loss of control

In the previous sections we have seen how the subsequent stages of technological development lead to an increasing and *inevitable* loss of control of the programmer over the machine. Increasing, because first, with declarative programming, is lost the control over the program flow. Next, with artificial neural networks, the programmer loses the symbolic representation of her program. All that remains to be inspected are a multitude of synaptic weights, which cannot be interpreted symbolically any more. The only way to measure the performance of a neural network (artificial or otherwise) is to *test* its reactions. Reinforcement learning brings about the loss of the computer's infallibility. Where up to now we could distinguish "correct" and "erroneous" programs, the technique of trial-and-error employed in reinforcement learning irrevocably blurs this distinction. Even the correct program cannot avoid errors, because the errors are the price for its adaptability. Genetic algorithms and genetic programming bring about the final loss of the code as well as of the training situation: instead of programming, the software engineer deploys his program's seeds as randomized virtual organisms in a simulated primordial soup, from which he hopes, in time, to see solutions emerge. He has never seen the code of these virtual organisms which represent his solutions, he has never programmed them. And autonomous agents, finally, are designed to act without supervision, so that an erroneous behaviour will be detected by the programmer only long after it has occurred and its consequences have been established, if at all.

Since it seems like the programmer is voluntarily giving away control over the machine, one might be tempted to talk of "abdication" instead of "loss" of control. But we should not overlook the fact that using advanced software design technologies is not an arbitrary decision of the programmer, but a necessity which is forced upon her by the constraints of the problem domain she has to tackle. Some highly dynamic and complex problems *cannot* be handled otherwise but by the use of neural networks and genetic algorithms. If the programmer wants to stay in business, she has no choice but to adopt these technologies. Thus, it might be said that the responsibility for adopting them lies with the society, the decision of which it is to demand that problem domains like street traffic control or automated diagnoses of diseases be handled by computer programs. Since society demands of the programmer to provide an automated solution, the programmer is arguably *forced* into the situation of reduced control, thus justifying us talking about *loss* instead of voluntary abdication of control.

The epistemic advantage of machines

Aside from the inevitability of error as part of the process of learning, modern computers often have another property that distinguishes them from the artifacts of the past: they have an epistemic advantage over their operator as well as their manufacturer, which renders effective supervision impossible.

Let's have a brief look at how supervision commonly works. If I supervise a child or a dog, I can do so effectively only if my faculties of understanding the world around me and my capability to predict the outcome of certain actions is more developed than that of the child or animal I am supervising. This *epistemic advantage* I have over the supervised makes me a suitable supervisor, and at the same time gives me a greater degree of control over the interactions of the supervised with his environment than he would have if left to his own devices. By *knowing more* than the child, I am in a position to *foresee* harm both to the child and to the environment, and thus to avert it. This higher degree of control also bestows on me a higher responsibility: commonly parents as well as dog owners are held responsible for the actions of their children and pets, as long as they have been supervising them at the moment the action occurred. This is the same pattern we have seen before: higher degree of control leads to higher degree of responsibility. The child's reduced control over the environment leads to it having to carry less (or no) responsibility for its actions.

A common misconception in the discussion about learning machines shows itself in the demand that they should be supervised more effectively, so that negative consequences of the machine's operation can be seen in advance and averted. In fact, such a surveillance of the learning automaton is, in many cases, impossible, because *it is the machine that as a rule has the epistemic advantage* over the human operator. Control systems of air planes and nuclear power plants, robots which assemble electronic parts with micrometre precision, deep space exploration devices, military self-targeting missiles, even internet search engines: these are only a few of the machines that cannot be controlled effectively by a human, because he is in a position of epistemic disadvantage: his senses don't permit him to measure the level of radioactivity inside a nuclear reactor, his speed of reaction is insufficient to control a modern fighter air plane, the precision of his eyes and fingers does not extend to micrometre level, and he has no sense at all for deep space radiation. Even something as simple as searching the internet cannot be done by a human any more. If I enter a search string into my favourite search engine, I will possibly get thousands of hits, and I will never be able to systematically check whether these hits really are the best available, and whether I was presented with the full list of hits or a censored or suboptimal subset. The epistemic advantage of machines is in these cases rooted in their superior processing power, in their possession of additional sensory equipment humans don't have, and in their almost unlimited and unfailing memory. Because of this, there is no way a human could outperform the controlling computer of a fighter plane or a nuclear power plant: he is a slave to their decisions, because he is physically unable to control and supervise them in real-time.

The responsibility gap

In the face of these developments, it doesn't seem meaningful (or just) to insist that a programmer should be responsible for an autonomous, learning program's actions in the same way that a car driver is responsible for the consequences of driving his car. If technological development leads to an inevitable loss of control, and if, as we have seen, such control is a prerequisite of just responsibility ascription, then we must either accept that *nobody* is responsible for the actions of some classes of learning automata; or we must, alternatively, find some other target to ascribe responsibility to. Or we must, and this would be the *ultima ratio,* dispense with modern technology, of the kind described above, altogether.

If we accept the existence of such a responsibility gap (Matthias, 2004a), we might ask whether the machines themselves might be suitable targets for responsibility ascription. Unfortunately, the philosophical discussion that has evolved around this topic has often been laden with confusion: confusion about the technical background of learning automata (and its implications) as described above, as well as confusion about what conditions are to be attached to the case of machine responsibility. Could a machine be legally liable? Under which conditions? Is the eligibility for moral agency a prerequisite to responsibility? Are the conditions for moral responsibility the same as for legal responsibility or different? Is personhood necessary for responsibility? Or even the plain and indispensable fact of being biologically human?

Can automata be ascribed responsibility?

The ladder to personhood

As long as we talk about humans, it is safe to use the concepts of responsibility, culpability, rehabilitation, punishment, intentionality and personhood which are based on anthropocentric views of what constitutes a (legal, moral, or social) person. It is only when we embark on the analysis of the non-human agent that we need to clarify these concepts and to point out the implicit anthropocentrisms contained in them. In Matthias (2004b) we proposed that common conditions and phenomena of both the legal and the moral spheres of interaction can each be ascribed to a specific type of agent for which they make sense. We distinguished the "legal person", the "moral person" (both of which should, in retrospect, have been named "agents" instead of "persons"), the "social person" and, finally, the biological human being, each with its own set of conditions that are to be met.

In the present context of responsibility ascription, what we really mean when we talk about machine responsibility is *legal* responsibility, or the ability to be *legally liable* for damages caused by the operation of the machine itself.

For our purpose of clarifying the legal liability of machine actions, it is not necessary to talk about moral agency or the possible metaphysical personhood of machines (for the difficulties of such an undertaking see, for example, Johnson, 2006). Our aim in the context of the present discussion is to solve the arising problems of responsibility ascription to machines inside the framework of civil law, and not to discuss retribution, resentment or punishment in relation to nonhumans. This would be a completely different field of inquiry, and one which is not likely to lead to usable, practical results any time soon. What concerns us are *damages* caused by the machine's operation: and these fall squarely into the realm of civil law, which operates mostly without metaphysical assumptions about the actors involved.

Now, to be legally responsible, one does not have to be human. Many legal systems have a concept of corporate entities (associations, companies, states) which can act and carry responsibility for their actions independently of the human beings that take part in the corporate structure. In international relations, for example, countries have rights and obligations that are different from those of the humans who represent them at any particular moment. It is crucial to see that the volition of the corporate entity, and thus the responsibility for its actions, is not at all derived from the volitions or the individual human-ness of its members. Often the members of a corporate entity (e.g. a stock corporation) are themselves not individual humans at all, but other corporate entities: companies or even governments. The volition of a corporate entity (for example of the government of a democratic country) cannot be derived from any mental states of any individual human agent (e.g. its prime minister); instead it is the product of a complex, rule-controlled process, which involves a majority of the individuals participating in the corporate entity (for example in the form of voting), and thus can only be attributed to the corporate system as a whole.

Still, every system which is to be ascribed a legally responsible status has to exhibit certain properties. Which exactly these are is not clear, since national laws differ significantly on this point, and, additionally, as a rule they don't bother to specify explicit conditions for legal responsibility. Instead, the borders between full, partial and absent responsibility are often deliberately left unspecified in order to be drawn individually for each case by the courts. A few points are, nevertheless, clear: a person under heavy drug influence is less responsible for his actions, as is someone who is, for medical reasons, unable to understand the complexity of a situation or its implications. In German law, persons can be conditionally able to assume responsibility for their actions: for example, compulsive gamblers can have reduced responsibility for their actions where gambling is concerned, but be fully responsible otherwise. If we try to locate specific conditions for (legal) responsibility inside the framework of contemporary analytical philosophy, we find five main points, which seem to be necessary as well as sufficient for it (Matthias, 2004b):

- **Intentionality** (Dennett, 1987), the requirement that an agent must be capable of intentional action, is clearly a prerequisite for responsibility. Thus a person is not held responsible for the consequences of involuntary movements, be it a reflex or an action undertaken while sleepwalking.

- Being **receptive and responsive to reasons** (Fischer&Ravizza 1993 and 1998) is another prerequisite for legal responsibility. Legally responsible agents must be receptive to reasons for or against a possible action, and they must respond to sufficient reasons against an intended action with a change in their plans and their subsequent behaviour. Rational understanding of causal relationships between actions and results, as well as the capability of the agent to act according to his understanding are basic requirements for legal responsibility in law.

- Having **second-order volitions** (Frankfurt 1971 and 1993). For example, in German law a person can be legally incapacitated "because he is in a pathological way controlled by the will of others" (Diederichsen 1984, 150). It is not only freedom *to act* which the law requires of a legally responsible agent, but the freedom *of the will,* the ability to choose the goals she desires to pursue. Only an agent who has the freedom to act according to her desires and the freedom to choose those desires freely, has all the freedom "that it is possible to desire or to conceive." (Frankfurt 1971, 16)

- Being **(legally) sane** (Wolf, 1987) means that the agent's second-order volitions must be "sane," that is, they must be compatible with the kinds of second-order volitions other agents in the same social context have. Effectively this requirement puts a third-order instance above the level of second-order volitions, thus exercising a social control over the choice of an agent's second-order volitions. Most legal systems recognize this requirement: an agent can be perfectly intentional and reasons-responsive and still be *insane;* in that his basic values system, which underlies all his individual choices, is not compatible with the basic values of the surrounding social context. Such an agent would generally not be held fully responsible for his actions.

- Being **able to distinguish between intended and merely foreseen** consequences of actions (Dworkin, 1987). Legal systems often distinguish "intended" from merely "foreseen" consequences of actions, and they presuppose the agent also to be able to do so; see, for example, the common as well as important distinction made between negligent and wilful homicide in most legal systems.

Though some of these properties have been proposed as being by themselves *sufficient* for (moral or legal) responsibility, we maintain that this is due to the implicit understanding that we are talking about humans, or at least systems that are partially similar or functionally analogous to humans. But now we are (in principle, at least, or as a thought experiment) able to construct artificial systems which exhibit some of those properties without the others. And thus we see, that they are each one of them *necessary* for full responsibility; but none of them is, in itself, *sufficient.*

Now, do machines satisfy these five criteria? Intentionality and reasons-responsiveness are exhibited (at least for limited domains of action) by every chess computer. Dennett (1987) has shown how the concept of intentionality can be separated from obscure metaphysical requirements (e.g. involving mental states, an approach that we find in Johnson (2006) and which is doomed to fail because of the complexity of the underlying mind-body discussion, which cannot be dismissed *en passant).* Expert systems have a long tradition of stating their reasons for particular choices. Also the concept of foreseen vs. intended consequences is one every planning component in an artificial intelligence system exhibits. More difficult to grasp technologically is the concept of *sanity,* because it presupposes a huge amount of knowledge about what kinds of volitions are commonly in use in a particular social context. Such knowledge will probably not be available before the problem of common-sense and everyday-knowledge for computers is successfully solved (for first steps in this direction see various publications of the CYC project, e.g. Lenat, 1995).

Social responsibility without personhood

In a very promising approach, Stahl (2006) tries to further simplify the problem of responsibility ascription to sub-personal agents, e.g. computers. He argues that the concept of responsibility needs not necessary be coupled with questions of agency or personhood, but can be seen as fulfilling a social role:

"An analysis of the concept of responsibility shows that it is a social construct of ascription which is only viable in certain social contexts and which serves particular social aims. If this is the main aspect of responsibility then the question whether computers can be responsible no longer hinges on the difficult problem of agency but on the possibly simpler question whether responsibility ascriptions to computers can fulfil social goals." (Stahl, 2006, 205).

He then goes on to propose a concept of "quasi-responsibility," which is analogous to (full, metaphysical) responsibility in its social function, but not confined to humans or persons alone. Of course, such a concept would be useless if it would not interface at some points to our real social environment, and thus be part of causal chains which involve "non-quasi" entities, and here lies one of the points that have still to be clarified: If I assign "quasi-responsibility" to a machine, and it causes harm or damage, this damage is obviously not a "quasi-damage". Accordingly, I cannot expect the machine to undertake only a "quasi-compensation" for the damage caused, or the legal system to provide me with "quasi-laws" to apply to the machine. But if we drop the prefix "quasi" from the consequences, then we must ask ourselves, how much exactly does "quasi-responsibility" actually differ from the "real" thing?

Future trends

Increasing use of learning automata in all areas of life will lead to a further widening of the responsibility gap. At the same time, increasing computing power widens the epistemic gap, thus rendering effective supervision of the machine even more unrealistic. Already, one particular internet search engine is responsible for much of the world's knowledge and information filtering, while at the same time, due to the amount of information processed, it cannot be effectively controlled by a human any more. Faster processing speed, coupled with increasing amounts of information, makes human evaluation of a machine's decisions (at least in real-time) impossible. At the same time, increasing computing power will enable us to deploy more and more products based on artificial neural net technology, which, by its very architecture, defies effective (symbolic) evaluation of the information stored within.

Already we are surrounded by autonomously deciding and acting technology. In areas like traffic control, automated disease diagnosis, automated control of flying vessels and submarines, as well as numerous others, we cannot but employ self-learning technology in ever increasing amounts, thus always giving away more and more of human control over the machines' decisions and actions.

As machines act autonomously and without effective supervision, we will observe an increase in cases where responsibility for the consequences of such actions cannot be justly assigned to a human. Up to a point, the resulting damage can be distributed to a broad base of a social group's members (for example, through unspecific insurance compensations or government intervention). But as such cases become more numerous, eventually a solution will have to be found which permits us to deal with them inside the regular conceptual framework of civil law (as opposed to ad-hoc compensations after the damage has occurred). It must be seen whether ascribing a limited kind of (quasi-) responsibility to the acting automaton itself may provide a solution to the problem which is both just and practical to handle in everyday legal transactions.

Conclusion

Some classes of learning, autonomous machines are artifacts *sui generis,* and they cannot be compared to conventional machines where the responsibility ascription for the consequences of their operation is concerned. There are various ways to deal with the resulting responsibility gap. If we want to avoid embarking on metaphysical discussions of machine personhood (which tend to involve aspects of the notoriously difficult philosophy of the mind-body-problem), we can limit the question at hand to legal issues (where non-human responsible entities are already present: corporations, states) and ask whether machines could be ascribed *legal* responsibility for their actions, leaving the moral issue aside. We find that legal responsibility requires at least intentionality, reasons responsiveness, second-order volitions, sanity and the ability to distinguish between foreseen and intended consequences of one's actions. All of those requirements seem not to be *in principle* out of the reach of future machines, but they are certainly not fulfilled at present. Another approach would be to assign "quasi-responsibility" to machines, a subset of "true" responsibility, as proposed by Stahl. It must be examined further what the exact merits of such a solution would be.

References

Adams, B., Breazeal, C., Brooks, R.A. & Scassellati, B. (2000). Humanoid Robots: A New Kind of Tool. *IEEE Intelligent Systems and Their Applications: Special Issue on Humanoid Robotics,* 15(4), 25-31.

De Jong, K.A. & Schultz, A.C. (1988). Using Experience-Based Learning in Game Playing. Proceedings of the Fifth International Machine Learning Conference, Ann Arbor, Michigan, June 12-14, 1988, 284-290.

Dennett, D.C. (1978a). Intentional Systems. In: *Brainstorms. Philosophical Essays on Mind and Psychology.* MIT Press, 3-22.

Dennett, D.C. (1978b). Conditions of Personhood. In: *Brainstorms. Philosophical Essays on Mind and Psychology.* MIT Press, 267-285.

Dennett, D.C. (1987). *The Intentional Stance.* Cambridge Mass., London: MIT Press.

Diederichsen, U. (1984). *Der Allgemeine Teil des Buergerlichen Gesetzbuches fuer Studienanfaenger.* 5th extended ed. Heidelberg: C.F.Mueller jur. Verlag.

Dworkin, G. (1987). Intention, Foreseeability, and Responsibility. In: Schoeman, F. (ed.) *Responsibility, Character, and the Emotions. New Essays in Moral Psychology.* Cambridge: Cambridge University Press, 338-354.

Fischer, J.M. & Ravizza, M. (1993). *Perspectives on Moral Responsibility.* Ithaca and London: Cornell University Press.

Fischer, J.M. & Ravizza, M. (1998). *Responsibility and Control. A Theory of Moral Responsibility.* Cambridge: Cambridge University Press.

Frankfurt, H. (1971). Freedom of the Will and the Concept of a Person, *Journal of Philosophy,* LXVIII, 5-21.

Frankfurt, H. (1993). Identification and Wholeheartedness. In: Fischer, J.M. & Ravizza, M. (eds.) *Perspectives on Moral Responsibility.* Cornell University Press, 170-187.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press: Ann Arbor.

Johnson, D.G. (2006). Computer systems: Moral entities but not moral agents. *Ethics and Information Technology,* 8, 195-204.

Lenat, D. (1995). CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM,* 38(11).

Llobet, E., Hines, E.L., Gardner, J.W. & Franco, S. (1999). Non-Destructive Banana Ripeness Determination Using a Neural Network-Based Electronic Nose. *Meas. Sci. Technol.,* 10, 538-548.

Luger, G. (2005). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* 5th ed. Addison-Wesley

Matthias, A. (2004a). The responsibility gap: Ascribing responsibility for the actions of learning automata. *Ethics and Information Technology,* 6(3), 175-183

Matthias, A. (2004b). Responsibility Ascription to Nonhumans. Climbing the Steps of the Personhood Ladder. In: Ikäheimo, H., Kotkavirta, J., Laitinen, A. & Lyyra, P. (eds.): *Personhood. Workshop papers of the Conference "Dimensions of Personhood"* (August 13-15, 2004). University of Jyväskylä, Publications in Philosophy 68.

Moriarty, D.E., Schultz, A.C & Grefenstette, J.J. (1999). Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research,* 11, 199--229

Sasaki, K., Markon, S. & Makagawa, M. (1996). Elevator Group Supervisory Control System Using Neural Networks. *Elevator World*, 1.

Schultz, A.C. (1991a). Using a Genetic Algorithm to Learn Strategies for Collision Avoidance and Local Navigation. *Proceedings of the Seventh International Symposium on Unmanned Untethered Submersible Technology.* University of New Hampshire Marine Systems Engineering Laboratory, September 23-25, 1991, 213-215.

Schultz A.C. (1991b). Using a Genetic Algorithm to Learn Strategies for Collision Avoidance and Local Navigation. *Proceedings of the Seventh International Symposium on Unmanned Untethered Submersible Technology.* University of New Hampshire Marine Systems Engineering Laboratory, September 23-25, 213-215.

Schultz, A.C. (1994). Learning Robot Behaviors Using Genetic Algorithms. *Proceedings of the International Symposium on Robotics and Manufacturing.* August 14-18, Washington DC.

Stahl, B.C. (2006). Responsible Computers? A Case for Ascribing Quasi-Responsibility to Computers Independent of Personhood or Agency. *Ethics and Information Technology,* 8, 205-213.

Stancliff, S.B. & Nechyba, M.C. (2000). *Learning to Fly: Modeling Human Control Strategies in an Aerial Vehicle*. Machine Intelligence Laboratory, Electrical and Computer Engineering, University of Florida. Retrieved June 3, 2007 from: http://www.mil.ufl.edu/publications.

Stolzmann, W., Butz, M.V., Hoffmann, J. & Goldberg, D.E. (2000). *First Cognitive Capabilities in the Anticipatory Classifier System.* Illinois Genetic Algorithms Laboratory Report No. 2000008. University of Illinois, Urbana.

Wolf, S. (1987). Sanity and the Metaphysics of Responsibility. In: Schoeman, F. (ed.) *Responsibility, Character, and the Emotions.* New Essays in Moral Psychology. Cambridge: Cambridge University Press, 46-62.

Zhang, B.T. & Seo, Y.W. (2001). Personalized Web-Document Filtering Using Reinforcement Learning. *Applied Artificial Intelligence*, 15(7), 665-685.

Key terms and definitions

Artificial neural network: A networked structure, modelled after a biological neural network, and implemented in software on a computer. Artificial neural networks enable computers to handle imperfect (noisy) data sets, which is essential for robust performance in advanced recognition and classification tasks (handwriting recognition, weather prediction, control of complex movements in robotic bodies).

Autonomous agents: Programs or programmed devices which act autonomously, without supervision of a human, often in a remote location (e.g. on a remote server or another planet). Since such agents are per definition required to operate without supervision, responsibility attribution for their actions to a human is especially difficult.

Declarative programming: A programming paradigm where the programmer does not specify the machine's behaviour in detail. Instead, she describes the problem to be solved in a kind of predicate logic calculus, leaving the details of the inference process to the machine.

Epistemic advantage: In general, the fact that an agent has by design better access to information about the world than another agent. In particular, the fact that some machines are in the privileged position to access data about the environment that humans cannot access (for example due to a lack of suitable sensor equipment, e.g. for gamma radiation or ultraviolet light); or that they are able to process information at a speed which transcends the speed of human thought, thus enabling them to handle situations in real-time, which humans cannot handle without machine aid (e.g. controlling a nuclear power plant, a low-flying fighter airplane, or a subtle orbital manoeuvre in space.)

Genetic programming: A programming paradigm where the program "evolves" as a string of symbols out of other strings of symbols. The "evolution" process mimics the mechanics of biological evolution, including operations like genetic cross-over, mutations, and selection of the "fittest" program variants.

Imperative programming: A programming paradigm where the programmer describes the machine's actions step by step, thus keeping full control over the machine's behaviour when executing the program.

Learning machine: A machine which modifies its behaviour after deployment, through adaptation to the environment in which it operates. Since its final behaviour at any moment depends not only on the initial programming, but also on the environment's inputs, it is in principle not predictable in advance.

Biographical sketch

After studying analytical philosophy in Göttingen, Germany, A. Matthias worked for fifteen years as a computer programmer, consultant, and head of the information systems group of the Computing Centre of the University of Kassel. During this time, he received his PhD in philosophy from the Humboldt University in Berlin with a thesis on "Machines as holders of rights". He presently teaches courses in both philosophy and programming languages at the University of Kassel, Germany.

Correspondence address

Andreas Matthias

Universität Kassel

Hochschulrechenzentrum

D-34109 Kassel

Germany