Code is Ethics — Formal Techniques for a Better World

Rolf Drechsler*† Christoph Lüth*†

*Cyber-Physical Systems, Deutsches Forschungszentrum für Künstliche Intelligenz
†Dept. Mathematics and Computer Science, University of Bremen
Email: drechsler@uni-bremen.de, christoph.lueth@dfki.de

***Abstract*—Computers are involved in our every-day life, making increasingly consequential decisions. This raises the question of the ethics of these decisions, for example when autonomous cars are concerned. We argue that the ethics of the decisions taken by a computer are in fact those of the developers, encoded in the program ("code is ethics"). This encoding is mostly implicit — programmers and users are often even not aware of the implicit decisions that are being made before the program is even run. We suggest that formal methods are an excellent way to make the criteria under which these decisions are taken explicit, because formal specifications are more concise, abstract and clearer than code, This way, it becomes clear why systems act the way they do, and where the responsibility for their behaviour lies.**

***Keywords*-ethics, formal methods**

I. Introduction

With computers pervading more and more aspects of our daily life, there is a growing concern about the influence computers have on our lifes. In the past, computers used to have a very simple and transparent functionality, but as systems and software grow more complex, they can produce results that are sometimes neither foreseen nor intended.

For example, when a bank is using software to administrate the accounts, there are no visible effects: one can only withdraw money that one has paid in to the account, or one may have an overdraft with usually substantial interest. For a large credit to finance a car or a house, a person such as the branch manager will decide whether that credit is given or not. If this bank is starting to use a credit scoring system to decide whether to give a credit or not, this decision now suddenly rests with "the computer": an algorithm seems to decide whether we can buy a new home or not, according to reasons which are unfathomable to us.

Another example are cars. Modern cars contain in an excess of hundred-fifty microprocessors controlling all aspects of the vehicle, but until recently such a modern car behaved to all intents and purposes like an older car without any embedded controllers: steering and velocity was controlled by the human driver. More recently, cars offer assistant control in clearly delineated situations such as manoeuvring in and out of parking spaces, detection of symptoms of sleepiness, or obstacle detection. In the not too distant future, more levels of autonomy may lead to fully autonomous cars.

In complex systems like these, the result of a computation (an algorithm) can have serious effects to consider:

- For example, the credit scoring might discriminate due to gender, postal code, or even ethnic background.
- And how would the autonomous car behave in situations like the "trolley problem" where it has to decide which action to take when all choices are harmful? [1]

If a human is confronted by these decisions, their decisions would be governed by their morality and ethics, including the regard for their fellow human beings, but what principles underlie the decisions made by an algorithm? The answer is that programs never make any ethical decisions by themselves. Programs *encode* ethical decisions, made by the developers during the design of the program. Unfortunately, these ethical decisions are usually implicit.

We suggest that formal methods are an adequate way to address this problem. Talking and reasoning about specifications and what they might mean makes the underlying assumptions explicit, as specifications can be far more concise than programs.

II. A Motivating Example

As a motivating example, consider a credit scoring algorithm which determines a credit rating. In its simplest form, the credit scoring operation might just take an account number, and return the maximal credit allowance. But from the account number, the bank in question may determine a lot of other input parameters: name, address, birthday, birth place, the history of transactions with the bank *etc.* Obviously, the bank has a legitimate interest to minimize the risks of defaults on the credits it pays out. On the other hand, the bank must not discriminate *e.g.* on the basis of gender or ethnicity when deciding whether to give a credit.

Research supported by BMBF grant SELFIE, grant no. 01IW16001.

[1]In the original trolley problem [1], a trolley with malfunctioning brakes is hurtling down a track on which five people are located which would be overrun and probably killed. The trolley can not be stopped, but we can operate a switch by which we can diverge the trolley onto a different track, where only one person would be affected. Would it be ethical to operate the switch? Many variations of the trolley problem exist, for example putting the lives of a group of elderly people against a single child; they are all concerned with the question of the ethics of the decision.

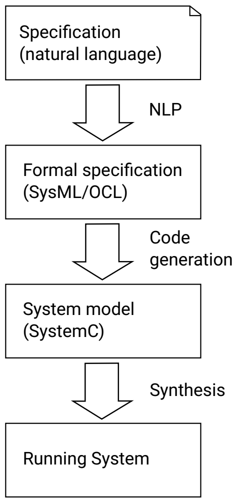
Specification (natural language)
NLP
Formal specification (SysML/OCL)
Code generation
System model (SystemC)
Synthesis
Running System

Figure 1. The SPECifIC design flow

To give a very basic formal specification of the credit scoring algorithm, let $\mathit{credit}(n, c)$ be an operation which takes two parameters — the name n and the postal code c — and returns the maximal credit allowance. Then to specify that the value of the second parameter (the postal code) is irrelevant, we write

$$\forall p, q.\, \mathit{credit}(n, p) = \mathit{credit}(n, q)$$

This specification, crucially, does not divulge the actual algorithm which is usually a trade secret of the bank in question, but is sufficient to guard against hidden discrimination based on where the applicant lives.

III. From Natural Language to Implementation

Contemporary formal methods can track requirements (and hence design decisions) from initial formulations in natural language down to the running system. In previous work [2][3], we have designed such a comprehensive tool flow for embedded systems.

Figure 1 shows the SPECifIC design flow. From a specification given in natural language, we create an initial formal specification (in SysML or UML), using standard natural-language processing (NLP) tools. The formal model contains classes modelling the system structure, with OCL specifications constraining the behaviour further. On this level, we can verify certain system properties, such as liveness (freedom from deadlocks) or consistency.

From the formal specification, an implementation on the system level can be derived, given in SystemC [4] (a system-level modelling language allowing hardware-software co-design). On this level, we can simulate the system's behaviour, allowing to test the system systematically.

In the last step, we can synthesize hardware and compile software to executable code to get the running system.

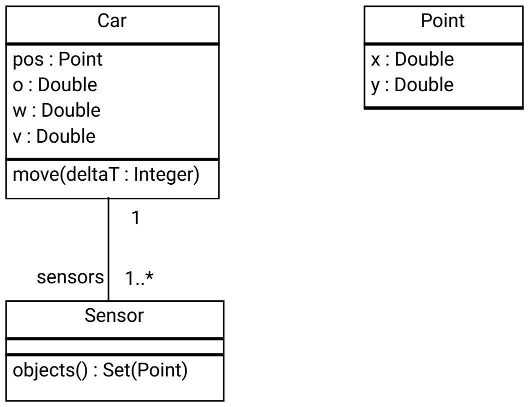

Figure 2. Modelling the obstacle detection of an autonomous car.

In each step of the process, we can track the entities from the levels above to the levels below. This way, we can precisely say which parts of the system pertain to a given initial natural-language specification.

IV. Autonomous Cars and Artificial Intelligence

To apply our formal methodology to the autonomous car example from above, we first need to give a formal specification of the car and its components, concentrating on the important aspects. Figure 2 models relevant parts of the obstacle detection of an autonomous car (using UML). The car itself has as attributes its current position (pos), orientation (o), steering angle (w) and velocity (v). From that, we can always calculate the current area, which consists of a set of points currently covered by the car. The car has one operation, *move*, which moves the car for a specified time interval (measured in milliseconds), using current steering angle and velocity. (We do not model how to change steering and velocity here.)

The car is equipped with a number of sensors, which are modelled by the class *Sensor*. A sensor returns a set of points, *objects*, which are points where it detects an obstacle. This is a useful and realistic abstraction from the way most sensors, in particular safety-directed ones, such as radar, lidar, or ultrasonic work.

We can formulate the safety predicate that the car never runs into obstacles: when moving, we never want the car to occupy points where obstacles had been detected. We can express this constraints in the object constraint language, OCL [5], as follows:

```
context Car:: move(deltaT: Integer) :
  post safe: area→intersection(
        sensors@pre.object()→ flatten())→ isEmpty()
```

This means that the area covered by the car after having moved is disjoint from the union of the set of obstacles from all sensors prior to moving, *i.e.* the car does not move into a space where obstacles have been detected.

From this, we can already learn a few things: firstly, if this postcondition is always fulfilled, the car will never run into obstacles, making the trolley problem vacuous. Secondly, this modelling is discrete, meaning the car jumps from one position to the next. While this is a useful approximation for small enough time units, for a precise characterization we should include the area covered while moving. (This is not entirely trivial, hence we omit it here, but see [6].) Thirdly, this only works if the obstacles do not change place between measuring and moving – in other words, if the obstacles are static. This is not really a realistic assumption, but if obstacles are allowed to move arbitrarily, the car can never be safe. So when refining this specification towards an implementation, we need to state precise restrictions on the movements of obstacles (*e.g.* an upper limit on the velocity) to maintain a safety invariant.

This leads to a discussion which obstacles we need to avoid. Specifically, some assumptions on human obstacles can be made (*e.g.* they will not move too fast, and will have a lower limit on the size); in fact, quite a number of such assumptions can be found in the safety literature, *e.g.* the required minimum diameter of a worker's leg (8 cm).

The modelling is very precise in the way obstacles are detected: as a set of points (a point cloud). This means we cannot really answer the trolley problem — the sensors would not even be able to distinguish five people from one. We can define a function which takes a set of points and returns a set of sets of points, clustering together sets of points which are physically close, in order to identify physical obstacles. Then we could implement an algorithm which tries to minimize the number of obstacles hit if collision is unavoidable. But to do so would be an ethical decision, and it would clearly be manifested in the specification. Crucially, this decision is taken by the *designer* of the system, not by the system itself.

Another aspect which can be seen here is that sensors as modelled do not have any semantic concept of what an obstacle is. Hence, a system which implements this model cannot distinguish between different kinds of obstacles, *e.g.* young children vs. older people. Again, to implement this requires a manifest design decision, and corresponding implementation effort.

V. What Now? Next Steps

An interesting next step would to connect the work on the SPECifIC design flow with ethics. There are a lot of ethical guidelines (such as the recent one from a high-level expert group commissioned by the European Commission [7]) which aim to assist programmers with the ethical decisions they need to make in their programs. As we have seen, these ethical decisions can be clarified and exhibited as formal specifications, but one might also use the NLP techniques to extract a formal representation of the ethical guidelines from the natural language text. If this is successful, we could use well-established formal method tools (model checking and theorem proving) to show that the decisions taken in the design of the program adhere to the ethical guidelines.

A program may be considered to be an act of communication; Knuth famously suggested we write programs in a way as to explain to other people what the program does [8]. An important aspect of this is that systems need to be able to explain, to the user, the designer's decision they are enacting. As we have argued above, formal specifications are a good abstraction of this explanation, and can serve as a good starting point for such *self-explaining systems*; first results on this topic can be found in [9], [10].

In closing, this paper has proposed that formal specifications are a good abstraction of the decisions the programmer has to take when designing the system, and in particular the ethical aspects of these decisions. When discussing these ethical aspects, formal methods serve to shift the focus from the system's behaviour (what is the computer doing) back to the designer's intent (what should the computer be doing).

References

[1] P. Foot, "The problem of abortion and the doctrine of the double effect," *Oxford Review*, vol. 5, 1967.

[2] R. Drechsler, M. Soeken, and R. Wille, "Formal specification level," in *Forum on Specification & Design Languages (FDL)*, 2012, pp. 37– 52.

[3] M. Ring, J. U. Stoppe, C. Lüth, and R. Drechsler, "Change impact analysis for hardware designs," in *Forum on Specification & Design Languages (FDL)*, 2016.

[4] D. Große and R. Drechsler, *Quality-Driven SystemC Design*. Springer, 2010.

[5] M. Gogolla and M. Richters, "Expressing UML Class Diagrams Properties with OCL," in *LNCS 2263*. Springer, 2002, pp. 85–114.

[6] H. Täubig, U. Frese, C. Hertzberg, C. Lüth, S. Mohr, E. Vorobev, and D. Walter, "Guaranteeing functional safety: Design for provability and computer-aided verification," *Autonomous Robots*, vol. 32, pp. 303–331, 2012.

[7] High-Level Expert Group on Artificial Intelligence, "Ethics guidelines for trustworthy AI," European Commission, 8. April 2019.

[8] D. E. Knuth, "Literate Programming," *The Computer Journal*, vol. 27, no. 2, pp. 97–111, 01 1984.

[9] R. Drechsler, C. Lüth, G. Fey, and T. Güneysu, "Towards self-explaining digital systems: A design methodology for the next generation," in *3rd International Verification and Security Workshop (IVSW)*, Costa Brava, Spain, 2018.

[10] G. Fey and R. Drechsler, "Self-explaining digital systems — some technical steps," in *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, Kaiserslautern, Germany, 2019.