Programs as tools for knowledge

Henri Salha

“Programs as tools for knowledge” may first evoke computer simulations, which philosophical importance is well known. But as one thinks a little bit further, other types of programs come to mind: for example, databases as they organize knowledge and enable search, counting, groupings, and correlations between large bodies of information; these in turn enable other more specialized programs, like tools for stylistic analysis in literature studies; and then, as other examples pop up, one starts to think that any program, as long as it manipulates data with a given knowledge value, could be considered as a tool for knowledge. A supply chain management software could be considered as holding critical knowledge for the company, such as inventory value, or pending customer orders, and so on.

The questions which are raised are therefore the following: 1) can we measure the “knowledge value” of a computer program? Databases, as long as they only store and retrieve information, seem to have a low knowledge value added; on the reverse, simulations, which mint their own data apparently from scratch, may hold much more added value; 2) how can we ascertain the epistemic link between a program, which is a “blind” symbolic process, and the portion of reality which it is supposed to represent? On this question, databases rely safely on the epistemic guarantee that is brought by the information they store, whereas simulations, as it is well known, have much more trouble to validate their link to reality.

These two questions are important for the philosophy of computing, as they may help to bring new perspectives in the understanding of the way computer programs can relate to reality. Further, they may change our understanding of what is knowledge itself, if it is a capability that computer programs may enhance or even hold. Cognitive sciences’ ambition to understand mental behaviors and knowledge in particular (as their name hints) through the paradigm of computing may be significantly rephrased when we understand how actual programs themselves are also vehicles and holders of knowledge.

In this communication, we try to demonstrate that three models of knowledge may answer to these questions, corresponding to three broad categories of software applications:

1) Functional applications, which basically work as processing data inputs into data outputs, give the traditional model where knowledge is understood as *coding*. Reality is attested as the source of the inputs – and knowledge is interpreted as a circuitry for combining and transforming them into meaningful representations. The power of this model is well known – as it encompasses activities such as calculation, counting, averaging, synthesis, search, comparison, inference and forecasting, to name some of the most obvious epistemic functions.
2) Processive applications, which actually interact with (and usually control) a portion of reality understood as a global process: think of a supply chain software, or avionics controlling key events in a airplane. Knowledge is here valued proportionally to its utility to this global process, and is validated by *feedback*, as pragmatists would have it. The added value of knowledge brought by these programs is usually low: their function is usually to keep information up-to-date, transmit it to the right users, or trigger key events accordingly, but not to *add* knowledge to the system. Some exceptions will however be illustrated.
3) Simulations and games, which execution is isolated from reality, and which link to reality has therefore to be entirely supported by an external observer to interpret it as reproducing, or schematizing, a portion of reality. This analogical model of knowledge requires external proof to hold, i.e. testing protocols. Against such a price, an important added knowledge can be sometimes derived, especially in the realm of complex systems which cannot be modeled by functional applications. Games add an interesting twist to simulations as they can be used to evaluate gamers themselves – e.g. in the case of educative software, read or write efficiency of the pupil, or in the case of a flight simulator, flying ability of an apprentice pilot.

We aim to show that this classification is systematic and therefore that these three models of knowledge are the basic ones, at least with regards to computer programs. We conclude by considerations on the implicit ontology that these epistemic models hint at.