What is a (logic) program?

Nicola Angius

In order to answer the question about what is a computer program from a logical point of view, one should focus on programs independently of their linguistic formulations. Indeed, computer programs can be high-level programs or low-level programs; low-level programs can be expressed in assembly language or in machine-code language; high-level programs can be expressed in many languages and, especially, they can be formulated through different programming language paradigms.

Indeed, contrasting ontological perspectives on the logical nature of programs come from distinct linguistic expressions thereof. For instance, Raymond Turner (2018) has it that high-level programs expressed by means of imperative languages can be defined as sets of operations upon machine states; programs should be rather considered mathematical functions if expressed in a functional programming language, or logical sentences when formulated through logic languages; finally, under the object-oriented paradigm, programs are sets of communicating objects. At the same time, given distinct formulations of a program into different programming language paradigms, and distinct compilations into different machine-code languages, one is dealing with the *same* program.

One way to grasp the logical structure underneath those diverse program formulations is to consider the *abstract machine* that any such formulation implements. A C program, a Java program, a Prolog program, are all distinct linguistic formulations, with distinct ontological commitments, of the "same program", the latter being expressible as the set of operations specified by the underlying abstract machine. *A program is logically equivalent to the set of operations it instructs.* Indeed, imperative, procedural, and object oriented programs make explicit reference to the machine states and operations on those states; logical and functional programming languages leave this duty to the compiler or interpreter.

According to this view, *a program can be logically defined as the equivalent class of implementations, expressed using different programming languages, of the same abstract machine.* An abstract machine specifies the implementing system's states, executions, variables, data types, data structures, algorithms, independently of any chosen programming language.

The present view on the nature of programs also accounts for low-level programs: assembly language and machine code programs belong to the equivalence class of programs implementing a given abstract machine. Indeed, both assembly language and machine code programs inherit the abstract machine implemented by some high-level language program.

This consideration leads to the conclusion that the ontological analysis of the nature of computer programs from a *logica*l perspective is connected to the analysis of programs from the *software systems* viewpoint. Given the abstraction level hierarchy defining software systems, answering to the question “what is a (logic) program?” amounts to identifying the level of abstraction that better grasps the (logical) nature of programs. Choosing, as one might expect, the high-level language program level is reducing, insofar as assembly language and machine code programs would be ruled out. By contrast, by focusing on higher levels of abstraction, such as the level of formal specifications, one would include too many entities in the equivalence class defined by the programs implementing the same specifications. Indeed, the same specification may be realized by different algorithms and consequently different programs which do not necessarily share the same set of states, operations, data types and structure.