Automatic Engineering Design Using Path Semantics

by Sven Nilsen, 2017

The use of path semantics can boost efficiency when designing systems for automatic engineering. One reason path semantics is powerful for this purpose is the ability to reason about a system at a higher level before implementing the lower level details. In common engineering disciplines a similar technique is used by e.g. using spreadsheets or building simulations. Path semantics can add more rigor to this process and increase understanding of the connection between the real world and models, or even various abstract levels of modeling.

Here is an overview of the techniques when using path semantics in automatic engineering:

- Abstract analysis
- Safety analysis
- Formalization
- Computable predictions
- Testing and verification
- Theorem proving
- Planning through programming
- Computer assisted reasoning

In path semantics a path is a functional relationship between functions using functions. It is a form that also has an equational representation:

f[g] <=> h

g(f(a, b)) = h(g(a), g(b))

The advantage of using path semantics is the ability to climb up and down a ladder that is traditionally associated with the “type” and “value” level of source code. In most programming languages, there are two different languages to describe types and values, but in path semantics there is just one language. The theory of path semantics describe other functional relationships used to check for consistency.

Path semantics help to formalize intuitive concepts as functions, and then you can proceed by reasoning about the functions in the theory of path semantics. This is where using and developing mathematical intuition helps solving problems and extending this ability to new areas. It also helps to make thinking more precise and writing down common sense assumptions. Since both questions and solutions are written in the notation of functions, one can measure the unknown and known and use the gained knowledge to compute stuff and make predictions.

The field of automatic engineering is extremely difficult. It requires a mixture of common sense modeling, constraint solving and artificial intelligence. Programming is often used as part of a creative process that goes far beyond our ability to analyze the complexity.

Tools that require dealing with the low level detail of programming objects are not efficient enough to be used in the field of automatic engineering. Path semantics deal with functions somewhere between the same high level as coding experiments and the low level required to fully formalize the behavior of a program. This give rise to an iterative cycle between ideas to be tested out as programs and the formal understanding of goals and the model of the problem.

Automated engineering is a challenging discipline because it aims at producing high productivity while also providing high security. This combination is often considered a trade-off in engineering. However, by leaving the heavy workloads to the computer and reuse earlier formalized concepts, one can reach further in shorter time because the foundation is more solid.

Path semantics offer analysis of a problem at a high level that is often much cheaper than performing a full analysis on a detailed low level. With gradually increased efforts over time and experience, one can obtain more formal advanced understanding of the domain.