Goals of path semantics

Path semantics is a sub-field of computer science and mathematics where you explore the space of meaning where functions are connected by functions, forming a natural hierarchy.

The mathematics of path semantics is derived from a single axiom. The axiom tells how symbols are interpreted at fundamental level. First you have atomic functions and arguments, but no return value. Applying the axiom a second time adds computation. Splitting the symbol of atomic functions with its arguments results in Type theory. Then one constructs Boolean algebra. Applying an infinite number of atomic functions gives the concept of numbers. This process keeps going, applying the axiom over and over, until all known concepts of mathematics are defined, or new mathematical concepts are found.

What makes path semantics special is that it has no limit on constructive creativity, as long as it never violate the original axiom. It builds on a notion of informal equality and acyclic order. There can be stronger or weaker forms of equality, and stronger and weaker forms of acyclic order. As we build more advanced languages and describe concepts formally, it turns out that differences in concepts follow exactly the meaning that is given by the axiom, because they are constructed from it. That is, one can study a concept from its theoretical description and find new and surprising properties that are true for the underlying system. It is a theory of patterns inside languages, about languages.

It is the job of a path semanticist to identify and collect information that relates various languages and semantical constructs, and decompose the meaning into formal constructs, or write down what is understood intuitively but can not be expressed in the working formal language. Path semantics is not governed by a single language, but using existing formal languages or constructing new ones for studying a problem.

The major goal is to develop meta knowledge about path semantics, sufficient for constructing computer programs with human level intuition to solve tasks such as mathematical exploration and organization of mathematical knowledge, model building by proof techniques from physical laws or by observing the world, and synthesizing of programs from specifications or conversation with humans.

This requires finding out what is reducible to the single axiom and what parts of mathematical intuition is hard to reduce, and how this distinction can naturally be automated in computer programs, either through training neural networks to learn the meta knowledge, or mapping the knowledge directly in formal languages that can automate some parts of the tasks.

General problems in mathematics and computer science

All mathematical concepts can be described in some language, but it is easier to describe them in some languages than others. All meta concepts can also be described in some language, but these are also easier to describe in some ways than others. We constantly invent new ways to use symbols, and this is how we can solve problems more efficiently and gain more understanding.

The goals of path semantics are roughly divided into two groups:

- Solve problems more efficiently
- Gain more understanding

Computers are way faster than humans, so solving problems more efficiently usually means figure out how to make computers do it instead of humans. The other goal, gain more understanding, is something that humans can do and benefit from. Even if computers can solve all our problems, it does not help if we do not have the wisdom to pass on our knowledge to the next generation. Perhaps some people out there are experts in something, but it does not help if you can not get in touch with them or hire them to help you out.

A very hard challenge that only can be worked on by a few

Society progresses slowly because concepts take time to learn, and not everyone learns the same concepts. Path semantics seeks a bottom up approach to bootstrap concept learning, but moving the dependency of concepts learned by society into computer software.

Path semantics is a challenging area of knowledge because it requires constructive learning to understand it. While most people learn something as facts, it is often not very deep. For example, learning to construct a language is a skill that is essential for understanding path semantics on a deep level, which is not taught by most schools. The skills of the practitioner supports the learning required to get new insights. Obtaining such skills in the first place can be a tough challenge. For example, you have to be intelligent in the first place to learn it, because without intelligence you can not figure out what you need to learn to reach the next goal.

Infer concepts from the chaotic environment

Since far from everything about path semantics is known, there is no environment you can construct where you can derive new concepts automatically. It often happens by playing around with some language, then look for patterns and test it. Once you have a pattern that seems to work, you try to derive new patterns and formulas that connects them. If the new patterns form a known concept, then you can infer a lot about the underlying language from things you already know. If the patterns form a new concept, then you study the new one and repeat the procedure until you get to a known concept.

There is a lot of overlap between other areas of mathematics and path semantics. One difference is that path semantics seeks insight into general knowledge about its own activity, with the hope that much of mathematics and science can be derived automatically by computers.

A major reason people working on path semantics study other areas is to get ideas of how these concepts can be creatively invented by a computer.

Another reason is that once you have a description of a system, you can learn the concepts that follows from its computationally structure. There are lot of areas that we want to describe mathematically to study and master. We want to know what kind of patterns are hiding inside the patterns. Even if we fail at creating these concepts creatively, we want to master the mathematics that is already described.

Path semantics as an engineering problem

Most of mathematics relies on equations for describing relations between variables. The problem is that semantics of equations is loosely coupled with the syntax. How do we know what to look for? This is a problem that path semantics solves. In path semantics, functions are connected to other functions through functions. When we have tested all functions that we have come across for a given function, there is nothing trivial left to learn and we can move on to study other functions. This corresponds to only a small fraction of the combinations that equations give, therefore narrowing down the target into meaningful statements. All path functions can be translated into an equation, therefore path semantics tells us indirectly which equations to look for.

Path semantics engineering is a discipline that tries to solve how to organize and represent the knowledge about functions such that it can be used by other computer programs. This is like building a knowledge base, but we are mostly interested in the mathematical kind. Algorithms that describe phenomena in the real world are interesting, but only if they are broad and general. The purpose with having such a knowledge base is to save work when doing new things, because the knowledge can be trusted to some known degree.

In the early phase, it does not have to be useful

A potential usage of path semantics is to construct programming languages which helps the programmer to write more efficient code. For example, a tool suggests to the programmer how to make non-trivial optimizations, simply by analyzing the code and applying path semantical knowledge.

Of course, it usually never turns out exactly how one visualizes the future of programming. Some other ideas that might catch on:

- Using proof tactics to improve a program without changing its meaning
- Make computers learn to understand what the source code means without running it
- Predict some aspects of the program
- Assist the programmer with making decisions

Another important aspect is that we do it because it is fun, or scratching an itch for knowing more, and reducing the confusion that we have about mathematics.

Programming and mathematics are activities that do not have to be useful. They can be abstract and art like in behavior, pursued just because somebody finds it beautiful. Thinking can be an art that results in equations or programs that behave like magic.