Philosophy of The Core Axiom

by Sven Nilsen, 2020

In this paper I discuss the mathematical philosophy of the core axiom of path semantics.

The core axiom of Path Semantics is the following:

$$\frac{F_0(X_0) \land F_1(X_1) \land F_0 > X_0 \land F_0 = F_1}{X_0 = X_1}$$

The syntax `F(X)` might be thought of as a kind of implication `F => X`,
or a predicate `F` returning `true` for some unique input `X`.

Although, it is sufficient to interpret the axiom using the implied meaning of the axiom,
since it models uniqueness up to the interpretation of equality and therefore also implication.

In first-order logic with equality `=` and declaration order `>`, a naive interpretation is:

$$\forall\ f, g, x, y \ \{ \ (f(x) \land g(y) \land f > x \land f = g) \Rightarrow (x = y) \ \}$$

This property contracts all inputs to a single equivalence class, such that the following seems possible:

$$(f(\text{false}) \land f(\text{true})) \Rightarrow (\text{false} = \text{true})$$

However, since `false = true` is `false`, one can reduce this to:

$$\neg(f(\text{false}) \land f(\text{true}))$$

What actually happens is that `f` can not be `\true`. The input must be unique.

When the inputs are unknown variables:

$$(f(x) \land f(y)) \Rightarrow (x = y)$$

This can be interpreted as putting `x` and `y` in the same equivalence class. The equivalence class is represented by `f`. If `f = g` then the equivalence class of `x = y` is represented by the equivalence class `f = g`. However, such equivalence classes do not exist on their own. They exist as a result of assocating symbols with each other. Therefore, supporting a Wittgenstein picture of mathematics.

As a result, the core axiom models a theory where equality becomes a technique for identification of associated meaning. This equality is the most basic form of logic, because it gives a method to construct mathematics while preserving the intuitive proof structure.

In more advanced interpretations using first-order logic, one can simply replace a single equality with multiple ones, without destroying the constructive properties of first-order logic in general.

Another form of the axiom, perhaps clearer for philosophy of mathematics:

$$\frac{F_0(\mathbf{X_0}) \wedge F_1(\mathbf{X_1}) \wedge F_0 > X_0}{(F_0 = F_1)(\mathbf{X_0 = X_1})}$$

Equality, in addition to just having a truth value, might be thought of as a higher order function that associates structurally one equality with another equality. With other words, the meaning associated with a symbol is identified with meaning of symbols identified with the symbol.

In the Wittgenstein picture of philosophy of language, the usage of symbols is what symbols mean. Therefore, the usage of symbols identified with the symbol is identified with the usage of the symbol.

One common source of misunderstanding path semantics is that the core axiom is modeled in a specific formal language, e.g. first-order logic. To understand why using a formal language is problematic, one must also develop some experience of implementing such theories. I will try to elaborate on this experience here and discuss the implications for the core axiom.

In any formal language where the semantics is not strictly computational (e.g. propositional logic), the semantics of proofs is structured around transformations of expressions.

For example, the for-all loop in first-order logic `∀` does not have any concrete meaning, despite the concrete meaning of analogues implemented in various programming languages (e.g. Dyon). In first-order logic, the `∀` loop is best understood as a syntactical expression with associated common usage. This common usage is that the meaning of the loop can be translated into a truth value `true` when each case of an enumerated collection is `true`. This is how people understand the `∀` loop.

However the intuition that people have is often approximations, not accurate modeling of semantics. It is possible that the meaning of something has a more precise semantics that implies the common usage.

From a path semantical perspective, the precise meaning of the `∀` loop in first-order logic is determined by the transformations that take place using its expression.

In general, transformations can be modeled using rewriting systems:

    <pattern> => <synthesis>

The left side describes a pattern that binds expressions to variables. The right side synthesizes the bound variables into a new expression.

When you try to formulate a foundation of mathematics, you want to avoid reference to any particular function, if possible. In Zermelo-Fraenkel Set Theory, the only function is the set membership function.

By limiting pattern matching of rewriting systems to generic functions, is it only possible to express rules that uses built-in special functions. For example, using logical NAND to model Boolean algebra.

This makes it possible to distinguish between data (input) and programs (rules). In the data, one can express e.g. `eq(a, a)` to say that `eq` returns `true` when both its arguments are `a`. In the rules, this would not work since `eq` would bind to any function.

In Zermelo-Fraenkel Set Theory, equality is defined in terms of the set membership function. One problem with this approach is the additional semantics that equality gets, from being defined this way.

The entire field of Homotopy Type Theory was developed because sets do not model equality itself. However, Homotopy Type Theory required modeling equality using Type Theory, which is a limitation. It is possible to model Type Theory using equality instead.

Path Semantics expresses equality in the core axiom directly, which might be thought of as the opposite approach of Homotopy Type Theory. Instead of modeling equality, it is assumed that it exists in some form. Instead of panicking and wonder what this *really means*, one can just take the equality in the simplest form which is the identity of atomic symbols, bootstrap into functions and take it from there.

The next thought that occurs, is why not limit the core axiom to atomic symbols? This is inappropriate because Path Semantics is much deeper than that. Path Semantics is an entire framework built around the idea of associating collections of symbols. This pattern occurs everywhere, at various scales, which eases the transition into e.g. isomorphisms and arbitrary equivalences.

In first-order logic, the semantics is grounded in what one can prove from an expression. However, in Path Semantics, the semantics is grounded in ways of associating expressions. The difference is that first-order logic shows how e.g. a function returns `true` for some input, while in path semantics the same is shown by feeding `true` to the existential path of the function, or providing an input such that the function returns `true`. As a result, path semantics has a clearer methodology of how to produce proofs, neatly organized by various associated expressions. It is more like working with blueprints for theorem proving, than reinventing the tools for each proof.

Equality is at the core of mathematics, but it is not the only issue. Another issue is the structure of associations. However, associations themselves have no meaning without equality.

The entire idea of the core axiom is to express unique associations, where the uniqueness is up to the equivalence classes of the chosen theory.

In first-order logic, one expresses that `f` returns `true` for a unique input, as following:

$$\exists!\ x\ \{\ f(x)\ \}$$

This is equal to:

$$\exists\ x\ \{\ f(x)\ \} \land \forall\ x, y\ \{\ (\ f(x) \land f(y)\ ) \Rightarrow (\ x = y\ )\ \}$$

Without some reference to equality `=`, it is impossible to model uniqueness in first-order logic. Even then, first-order logic does not model actual functions, but merely some properties of functions.

Who can then blame Path Semantics for using equality in the core axiom?

Another question is whether Path Semantics does essentially the same as first-order logic seen from the perspective of the core axiom. The core axiom can not construct functions directly. However, it provides just enough semantics to start constructing functions using bootstrapping. In first-order logic, it is common to model properties of functions directly, instead of using bootstrapping. Path Semantics justifies the construction of functions, while first-order logic justifies the proofs about functions.