Comments on Leibniz's Law

by Sven Nilsen, 2018

A particular interesting semantics regarding identity of objects has come to my mind lately. These ideas are still being processed, but I will attempt to express some of them here.

Leibniz's law:

$$\forall\ x, y: A\ \{\ (\ x = y\ ) = \forall\ P: A \to \text{bool}\ \{\ P(x) = P(y)\ \}\ \}$$

Two objects are equal in every aspect if and only if every predicate taking them as arguments returns same truth value.

One reason I believe Leibniz's law is important is because a predicate carries the proof of whether two objects are distinct. If any predicate returns a different truth value, then there is no need to check the other predicates, we know that the objects are not the same. This is useful because in the real world we rarely have all the information about objects, but we only know some aspects of them.

On the other hand, in foundations of mathematics it is common to use sets, which equality requires comparing every member against all members of the other set. While sets in foundational mathematics are not intended to be used in practical applications, this leads to a contradiction because if all of mathematics can be modelled with sets, then every object can be thought of as a set. With other words, the real world might be replaced by a simulated version using only sets, and this simulation would be indistinguishable from the real world. The problem happens when you have some agent trying to model the world. In the real world, using set equality as identity is computationally intractible for modeling.

However, by exploiting Leibniz's law, an agent can perform proof operations without needing access to all information about an object. Think of this as two people, Alice and Bob, where Bob sends a description of some object to Alice. Alice can determine whether Bob is looking at another object when the same sentence has another value, such as "this object is red" and "this object is blue" implies that Bob is talking about two different objects (unless the object can be both red and blue).

Similarly, modeling the world requires some sort of language usage. The language is used to abstract away the need for immediate paying attention to the world, such that the agent can reason using its extracted knowledge with sentences as reliable information about objects in the real world. In this context, Leibniz's law is more important than explicit equality operations, because one only needs a few predicates to start reasoning about the world.

I think it is interesting to divide predicates up into families of functions. Each family of functions represents a language. Two objects might then be identical according to some language `A` but distinct according to some language `B`. Languages are ways of talking about objects.

$$\text{identical} := \backslash(L : \text{language}, x, y) = \forall\ x, y\ \{\ (\ x = y\ ) = \forall\ P: L\ \{\ P(x) = P(y)\ \}\ \}$$

identical(A, x, y)	`x` and `y` are identical according to language `A`
¬identical(B, x, y)	`x` and `y` are not identical according to language `B`

Consider an atomic function with no arguments. This kind of object can be thought of a computation where some object `a` is replaced by some object `b`.

a → b `a` is replaced by `b`
b → c `b` is replaced by `c`

Atomic functions with no arguments are very simple objects. This means that there are not many predicates that says something about them, without repeating what other predicates say.

Imagine that we have a function `returns` that tells what atomic symbol is replaced with another:

returns : expr → expr

returns(a) = b
returns(b) = c

This function is a complete description of a system of atomic functions with no arguments. In path semantical notation, one can define some sub-types:

a : [returns] b
b : [returns] c

Assume that there is some unknown symbol `x` that is either `a` or `b`:

x : [returns] c

Since `b` is the only symbol that returns `c`, then `x` must be `b`.

Using `returns` as the single function of a language $`L_0`$, that is, a family of functions containing only one member, it is not sufficient to construct predicates. One needs `b` and `c` too:

L_0 := { [returns] b, [returns] c }

As a shorthand version, one can define a higher order family of functions using functions alone, where the predicates are constructed from the existential path constrained to the trivial path:

x : [∃returns{∀returns}] true

L_0 := { [returns] x_0, [returns] x_1, … }

This means that the values that the function returns given all possible inputs, are the arguments of the sub-types which are used to construct all predicates, limited to the language of that function.

In general for multiple functions, taking the union of sub-types:

language := \(fs : T → any) = ∪ f : fs, x : [∃f{∀f}] true { [f] x }

identical(language([returns]), x, y) `x` and `y` are identical according to `returns`

The motivation of using atomic functions is to study formally how identity works for very simple languages.

One idea I have in mind is to express one object is constructed from simpler object, such as normal functions from atomic functions, and thereby prove that the language of atomic functions can not talk about normal functions. With other words, no normal function object is identical to any object talked about in the language of atomic functions. If a such thing can be proven, then it might also proven that it is always possible to create new ways of talking about objects, by applying Cantor's higher cardinalities of infinity.

Another future achievement would be to prove formally that Zen Rationality can talk about intelligent agents that are not possible to talk about within the framework of Instrumental Rationality, because Zen Rationality is an extension of Instrumental Rationality with fewer assumptions. This involves two new ideas, one that some philosophical concept can be used as a language, and another that making fewer assumptions carries a proof of set-like relationships between those concepts.

A long term achievement would be to create some sort of algorithm that is capable of starting with a very simple language, such as the one of atomic functions, and extend its own capability with more complex languages. The longer this algorithm runs, the more complex languages it will develop. Like a SAT-solver capable of finding objects satisfying boolean constraints, the algorithm should also be able to construct objects that satisfies the predicates that the language talks about.

These ideas are the ones I wanted to express related to Leibniz's law. The future will show whether the achievements are feasible.