Uncertain Identity Modality

by Sven Nilsen, 2018

The whole of path semantics is built on the idea that identity means you can say the same things. Likewise, if you can say different things it means some stuff are not identical. In this paper I propose an idea that might seem inconsistent with the concept of identity at first sight, because it appears to violate common sense. This is done by introducing a modality which allow special rules when an identity is unknown. I show that uncertain identity modality is not in conflict with path semantics.

The basic situation where uncertain identity modality is used, is when someone measures a variable that takes on a discrete number of states. For example, if `x` can be either `1`, `2` or `3` then it surely must be in one of these states. Notice that I did not say that `x` can be in more than one state, such as in quantum mechanics, but that `x` is in one state by definition.

It seems obvious that you then can think hypothetically about `x` as being in either `1`, `2` or `3` and assume this is everything that can be said about `x`. Yet, the concept of uncertain identity modality implies that this is not true. There is more to be said about `x`!

Let us say you create a scenario for each possible value, which separately is checked against some consistency criteria. No matter what value `x` might have, you could execute some scenario selected e.g. by how likely you believe `x` having a particular value.

An uncertain identity modality is the concept that even though you can construct a complete set of scenarios, it does not justify executing any particular scenario, in the absence of knowledge about which value `x` has. More, this modality makes it possible to assign a new scenario.

This can be argued for in the following way: Suppose that one knew which value `x` had and then erased the memory. First, one would have a reason to execute a particular scenario consistent with the value observed. Next, since one has no longer the knowledge about the value of `x`, one has no reason to execute any particular scenario besides taking some risk that the scenario will be inconsistent. Therefore, a particular scenario can not be justified without knowing which value `x` has.

The rest of the argument is constructed as following: Which scenario that is consistent to play out is neither one of the scenarios assigned to any particular value of `x`, because one can say that it differs in justifiability. According to path semantics, this means that the new scenario is not identical and therefore different. Surprisingly, this also holds if all scenarios are the same for any particular value.

It is far from obvious that some scenario must be played out given the uncertainty of `x`. However, assume that another value `y` is determined. The variable `y` takes on value `1`, `2` and `3` like `x`. No matter what thought process one uses under uncertainty, eventually `y` must be assigned a value. This means that the kind of scenario under uncertainty is similar to the scenarios where `x` has a particular value. It has the type `X → Y`.

The trick is to treat the type `X → Y` not as an unknown function, but as a family of functions. The family of functions where `x` is of a particular value has size 1 and corresponds to a particular function. When introducing an uncertain identity modality, the size of the function family can be larger than 1.

This means that in order to plan for every possible scenario, one must also plan for scenarios where there are unknown variable states. In practice this means treating `x` as if it has an `unknown` state:

$$x' \in \{1, 2, 3, \text{unknown}\}$$

This works because two functions from the function space `X → Y` can be projected with a product:

$$x_0' \in (\{1, 2, 3\}, \text{known/unknown})$$
$$y \in \{1, 2, 3\}$$

One can imagine the function space `$X_0$ → Y` encoded like this:

known	unknown	
111	111	
112	111	
113	111	
121	111	
122	111	
123	111	
131	111	
132	111	
133	111	
211	111	
212	111	
213	111	
221	111	
222	111	
223	111	
231	111	
232	111	
233	111	
311	111	
312	111	
313	111	
321	111	
322	111	
323	111	
331	111	
332	111	
333	111	
111	112	This is inconsistent because one can not differentiate between `1`, `2` and `3` when the variable is unknown
…	…	

When the variable is unknown, the input arguments of particular values of `x` must be irrelevant, so the size of the constrained function space is `$27 \cdot 3 = 81$`. This is the same size as adding another member:

$$1111$$
$$1121$$
$$1131$$
$$\dots$$

$$3^3 \cdot 3 = 3^4$$

Which means that the following type extension is valid:

$$x' \in \{1, 2, 3, \text{unknown}\}$$

In Rust this can be done with `Option<X> → Y`.