Witness in Path Semantical Logic

by Sven Nilsen, 2020

In this paper I explain why a witness is required for one-to-many associated propositions.

Path Semantical Logical separates propositions[1] into levels, such that an equality between two propositions in level N+1, propagates into equality between uniquely associated propositions in level N. For example, if `f` has level N+1 and `x` has level `N`, then `f(x)` associates `x` uniquely with `f`.

A proposition in Path Semantical Logic can be thought of as a symbol. One is able to associate symbols with something. The identification of symbols leads to identification of what the symbols mean.

When two symbols `f` and `g` are equal, their associated meaning is also equal:

f(x), g(y), f=g => x=y

Since `f=f`, one might think that the following is true in Path Semantical Logic:

f(x), f(y) => x=y

However, this is not the case!

The reason for this is that propositions have their own particular semantics, that can be a bit tricky. In order to explain this, it is simpler to start with another example.

The following is a tautology in propositional logic:

(f => x) ∧ (f => y) => (f => x=y)

Path Semantical Logic uses implication `=>` under the hood, so one could write this as:

f(x), f(y) => f(x=y)

Notice that this would be invalid syntax in typed first-order logic.
In typed first-order logic, if `x : T` and `y : T`, then `x=y : T=T.
The predicate `f : T → 𝔹` can not take `T=T` as argument.

In Sized Type Theory[2], this type error is exploited, overloading with `f(x~=y) = (f(x)~=f(y))`.
However, in general, `f(x=y)` would assume that `f : 𝔹 → 𝔹`.

Now, it happens that since all types in propositional logic are `𝔹`, could one make this work?
There is a problem though: For all `x` and `y`, it is not true that `(x=y)=x`.

(x=y)=x Is **not** a tautology!

Think about it: If `f(x)` and `f(x=y)`, then all inputs to `f` are the same, then `(x=y)=x`.

This means that if the following was true in Path Semantical Logic:

f(x), f(x=y) => (x=y)=x

Then the following would also be true:

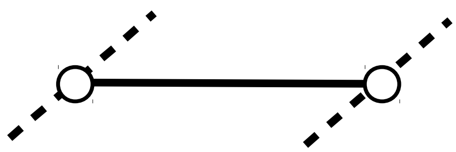
f(x), f(y) => x=y   Replacing `x=y` with `y` to get `f(x), f(y) => y=x` and change `y=x` to `x=y`.

However, since the first is not natural, the second is also not natural.

Yet, there is a way out of this problem: Introduce a witness.

To understand witnesses better, one might think of `f(x)` and `f(y)` as two points in a space.

The space `f` is contractible if there is a line connecting any two points.
A witness of the line is a surface that intersects with the line:



Instead of allowing a space to be contractible for any line,
one can require the existence of a surface intersecting with the line.
Any surface will suffice, it just needs to be mentioned explicitly.

However, instead of doing this for every pair of points in space, there is a shortcut:
Connect two spaces `f` and `g` directly, using `f=g`.

In Path Semantical Logic, one can prove the following:

f(x), f(y), f=g => x=y

However, one can **not** prove the following:

f(x), f(**x=y**), f=g => x=(**x=y**)   Substituting `y` with `x=y`

With other words, substituting a variable `y` with `x=y` is not sound in Path Semantical Logic.

Yet, one can prove the following:

f(x), f(x=y), f=g => x=y

As if by magic, Path Semantical Logic seems to know how to destructure equality.

The proofs in this paper were checked by an implementation[3] of Path Semantical Logic.

References:

[1] "Propositional calculus"
Wikipedia
https://en.wikipedia.org/wiki/Propositional_calculus

[2] "Sized Type Theory"
Sven Nilsen, 2020
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/sized-type-theory.pdf

[3] "Faster Brute Force Proofs"
Sven Nilsen, 2020
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/faster-brute-force-proofs.pdf