More About Witness Duality

by Sven Nilsen, 2020

The Product Witness[1] has its origin in equivalence as a binary relation.

An equivalence relation[2] consists of 3 properties:

- For every object `a`, `a = a` (reflexivity)
- If `a = b` then `b = a` (symmetry)
- If `a = b` and `b = c` then `a = c` (transitivity)

Notice that symmetry and transitivity properties are conditional, while reflexivity is without conditions. This means that reflexivity is a global property while symmetry and transitivity can be local. With other words, reflexivity holds for every object, while equivalence might hold for some objects but not all.

The Loop Witness[1] has its origin in equivalence as an identity morphism.

An identity morphism comes from Category Theory[3]:

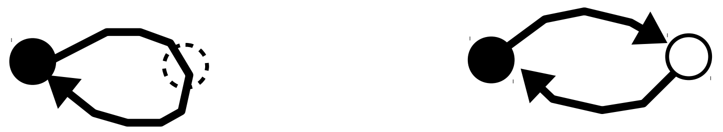
- For every object \`X\` in a category, there exists a morphism \`$\mathrm{id}_X : X \to X$\` (identity)
- For every morphism \`f : X → Y\` and \`g : Y → Z\`,
  there exists a morphism \`g · f : X → Z\` (composition)

Notice that composition in Category Theory is similar to transitivity in an equivalence relation.
In that sense, the identity morphism looks similar to the reflexivity property.

A category lacks the analogue of the symmetry property.
When adding the analogue of symmetry to Category Theory, one gets a groupoid[4]:

- For every morphism \`f : X → Y\` there exists a morphism \`$f^{-1} : Y \to X$\`

This groupoid property might be thought of as a 1-avatar extension[5] of identity morphisms:



With other words, starting with a *core* (black) with an identity morphism, one inserts a *witness* (white).

A groupoid requires this property to hold globally, but when working with identity morphisms, one can think about this as a local property within a category plus avatar extensions. In some sense, this makes the category being a partial groupoid locally. It is only valid to perform this operation on identity morphisms, because equivalence in this context is not general for all morphisms.

Similarly, an equivalence relation can operate on objects with other properties, e.g. `a => b` (logic).

For an equivalence relation, the transitivity property is only defined for equivalences.
However, composition in Category Theory is not just defined for identity morphisms.
Yet, since composition holds for all morphisms, it also holds for identity morphisms.
One can think about it as Category Theory with avatar extensions on identity morphisms,
as being a slightly stronger theory than equivalence relations.

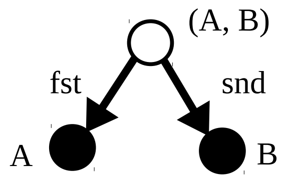
The missing property of equivalence relation has an analogue in logic:

- If `a => b` and `b => c`, then `a => c`

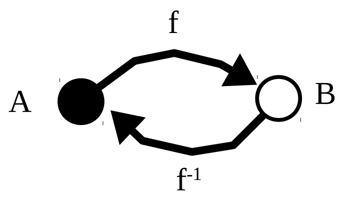
Just ignore this detail for now, it is not relevant.

Now, think about equivalence relation and avatar extensions on identity morphisms as different models of the same idea (equivalence), but with different expressiveness of language.

The equivalence relation view is associated with a Product Witness,
which might be thought of as some computation of a Cartesian product `(A, B) : [eqv] true`:



The identity morphism view is associated with a Loop Witness,
which might be thought of a proof of an isomorphism `$f \cdot f^{-1}$ <=> $id_A$` and `$f^{-1} \cdot f$ <=> $id_B$`:



Here, avatar extensions introduces a map `f` with an inverse `$f^{-1}$`, projected onto the identity morphism.

The Loop Witness allows transporting from `A` to `B` and vice versa.

There is no analogue transport for the Product Witness:

(A, X) : [eqv] true

This does not tell us what `X` is, only that there must exist some way of transporting `A` to `X`.
The Product Witness lacks the ability to introduce isomorphisms as map projections onto equivalence.

However, the Loop Witness is weaker in the sense that one can not express the existence of a way of transporting from `A` to `B`. Instead, the only way of expressing transport is to provide a transport.

The general picture is that Product Witness talks about the same idea (equivalence) as Loop Witness, but neither can express fully what that idea is. One needs both of them to understand equivalence fully.

To distinguish between Product Witness and Loop Witness, one introduces two abstract operators:

- `A = B` means `A` and `B` are related by a product witness
- `A ~= B` means `A` and `B` are related by a loop witness

The properties of equivalence holds for both operators.

Normally, `=` is used for equality and `~=` is used for the existence of an isomorphism.
Witness Duality does not make this distinction, because both might describe e.g. isomorphisms.

In Witness Duality[1] for Type Theory[6], one uses `=` restricted to types, while `~=` has 3 variants:

	Type A	**Value (a : A)**
Type B	A = B	a ~= B
Value (b : B)	A ~= b	a ~= b

An isomorphism `a ~= b` between `a` and `b` has the type `A = B`:

(a ~= b) : (A = B)

The isomorphism `A ~= b` can be thought of as a sub-type or fibration[7]:

a : [f] b A ~= b

$\because$ f : A → B

The isomorphism `a ~= B` can be thought of as a sub-type or fibration:

b : [f^{-1}] a B ~= a

$\because$ f^{-1} : B → A

The analogue of the Univalence Axiom[8] can be interpreted for Witness Duality:

(A = B) ~= (A ~= B)

For Type Theory, `A = B` is interpreted as existing only in the type level (relative to value level).
`A ~= B` is interpreted as existing in the value level, since it can distinguish between isomorphisms.
Univalence means that the witness of Witness Duality between type levels is a Loop Witness (`~=`).
Therefore, there are 3 variants of the Loop Witness, but only 1 variant of the Product Witness.

The rationale for this property is simply transitivity:

if `a ~= b` and `b ~= c` then `a ~= c`

The 3 variants of `~=` work through the type level and across type levels, propagating equivalence.
This is closely related to the core axiom[9] of Path Semantics.

There are two consistent type theories which satisfy the core axiom of Path Semantics:

1. Types as propositions, programs as proofs (Curry-Howard correspondence[10])
2. Arbitrary equivalences between programs, equality between types as existence of equivalences

This is known as the "Duality in Path Semantical Logic"[11].

In Path Semantical Logic, one gets the first theory by putting types in level 1 and values in level 0.

a : A A(a)

The second theory puts types in level 0 and values in level 1:

a : A a(A)

In the first theory, if `a : A, b : B, A = B` then `a ~= b`.
This means, there exists a way to transport from `a` to `b` when `A` and `B` are equal.
It holds for every member of `A` and `B`.

In the second theory, if `a : A, b : B, a ~= b` then `A = B`.
This means, when there exists a way to transport from `a` to `b`, the existence of this transport is provable using `A = B`. It does not mean that one can transport from every member of `A` to every member of `B`, only that there exists *some* way of transporting from `A` to `B`.

In the first theory, `a ~= b` is provable from `A = B`, which satisfies univalence, but collapses values.
Isomorphisms propagate automatically between every pair of members of the two spaces.

In the second theory, `A = B` is provable from `a ~= b`.
Isomorphisms propagate through transitivity, but not automatically.
This theory does not satisfy the Univalence Axiom, interpreted as for Witness Duality.

These two theories are the cocategory[12] of each other, where morphisms are type memberships.

The analogue of the Univalence Axiom for Witness Duality says that type membership must be of a special kind in order to unify these two theories. When this special type membership is distinguished from normal type membership, one can propagate isomorphisms within these constraints automatically, which is some sort of proof of transitivity. This special type membership can be parameterised.

In Cubical Type Theory[13], one solves this problem by introducing a `Path` type.
Normally, a heterogenous path type `PathP` is introduced, which allows defining a homogenous `Path`:

```
Path (A : U) (a b : A) : U = PathP (<_> A) a b          (source: cubicaltt[14])
```

The Path type is inhabited by maps from the unit interval, with endpoints `0` and `1`, to some type:

```
p @ 0 <=> a
p @ 1 <=> b
```

The geometric intuition is that the path type creates lines out of points, parameterised by unit interval.

References:

[1] "Witness Duality"
Sven Nilsen, 2020
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/witness-duality.pdf

[2] "Equivalence relation"
Wikipedia
https://en.wikipedia.org/wiki/Equivalence_relation

[3] "Category theory"
Wikipedia
https://en.wikipedia.org/wiki/Category_theory

[4] "Groupoid"
Wikipedia
https://en.wikipedia.org/wiki/Groupoid

[5] "Avatar Extensions"
AdvancedResearch – Reading Sequence on Path Semantics
https://github.com/advancedresearch/path_semantics/blob/master/sequences.md#avatar-extensions

[6] "Type theory"
Wikipedia
https://en.wikipedia.org/wiki/Type_theory

[7] "Fibration"
Wikipedia
https://en.wikipedia.org/wiki/Fibration

[8] "Homotopy type theory - The univalence axiom"
Wikipedia
https://en.wikipedia.org/wiki/Homotopy_type_theory#The_univalence_axiom

[9] "Path Semantics"
Sven Nilsen, 2016-2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/path-semantics.pdf

[10] "Curry-Howard correspondence"
Wikipedia
https://en.wikipedia.org/wiki/Curry%E2%80%93Howard_correspondence

[11] "Duality in Path Semantical Logic"
Sven Nilsen, 2020
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/duality-in-path-semantical-logic.pdf

[12] “cocategory”
nLab
https://ncatlab.org/nlab/show/cocategory

[13] “cubical type theory”
nLab
https://ncatlab.org/nlab/show/cubical+type+theory

[14] “cubicaltt”
Experimental implementation of Cubical Type Theory
https://github.com/mortberg/cubicaltt