Examples of Sized Type Equivalence

by Sven Nilsen, 2020

Equivalences in Sized Type Theory are not as intuitive compared to dealing directly with equality or isomorphisms. An equivalence means “there exists an isomorphism” without providing direct evidence. It is a technique of talking about isomorphisms without mentioning a specific identity. Equivalences are vague by design, on purpose.

Therefore, it can be hard to wrap your head around what equivalences are and how they work. When working with equalities or isomorphisms, you are thinking about them as functions taking some input and returning some output. This is not the case with equivalences. An equivalence is more like associating two arbitrary elements with each other. Just like a tuple:

(1, 2) : (nat, nat)	tuple
1 ~= 2 : nat ~= nat	equivalence

This breaks the intuition of equality, because when you see \`=\`, that is what you are naturally thinking. Notice that equality is not equivalence. Every equality is an equivalence, but not the other way around.

In fact, you can create an equivalence out of any two natural numbers, not restricted to equal numbers. There exists an algorithm to map back and forward between any numbers, so it is a valid equivalence.

Why not just use tuples? You could, but equivalences has some benefits as distinct notation:

- Function application permits transport of equivalences (even if there is a type mismatch)
- An equivalence can only be constructed if the terms are of same size (using Sized Type Theory)

For example, you can have \`1 ~= 2\`, but you can also have \`bool ~= bool\`.
This is because \`|1| == |2| == 1\` and \`|bool| == |bool| == 2\` in Sized Type Theory.

An equivalence is kind of like a choice, like introducing a new axiom, although there are rules and restrictions on what these axioms can do. Just look at the following example to see what I mean:

```
income(“john” ~= “peter”)
income(“john”) ~= income(“peter”)
1000 ~= 2000
```

If John earns 1000 and Peter earns 2000, then John’s 1000 is equivalent to Peter’s 2000, when John and Peter are equivalent with respect to income. This sounds a bit like comparing apples and oranges, or perhaps like transferring from one currency to another.

Equivalence is a way to choose freely what to associate with each other and reasoning about what happens when a specific choice is focused on. The choice is always available and valid, yet it is hard to think about choices that are not made otherwise. It is like expressing this relation brings it into attention and this is the role of equivalences as a language building block. Very subtle, indeed.

Function application of equivalences is what gives context to equivalences. Since you can use equivalences with any function, it does not get very interesting to think about this in general, until the moment when you pick one function to use. The very nature of this function determines e.g. whether a proof is valid.

In the previous example, the function `income` is not necessarily invertible, which is the same as not being an isomorphism. In that case, it did not matter, because one wanted to talk about something equivalent, which was two different incomes, in a non-invertible way compared to something more complex, which was John and Peter.

In other cases you want to be sure that function application does not destroy the structure you started with. For example, you can solve equations like this (using notation for algebra):

$(- 3)(x + 3 \sim= y)$
$(- 3)(x + 3) \sim= (- 3)(y)$
$(x + 3) – 3 \sim= y – 3$
$x + (3 – 3) \sim= y – 3$
$x + 0 \sim= y – 3$
$x \sim= y – 3$

In order for this method to be valid, the functions must be have an inverse, otherwise one can not be sure that the output talks about the input in the way intended.

The reason one can use equivalences to solve equations is because every equality is an equivalence.

No special rule was needed to apply the function to both sides of an equation.
This semantics follows from the rules of equivalence in Sized Type Theory.