Natural Equality

by Sven Nilsen, 2017

Path semantics requires a notion of equality, but exactly *how* does this equality work? Is there a natural interpretation of equality, that must hold for all path semantics?

In normal programming languages there is a loose notion of equality that define it simply as a binary operator that can be overloaded for certain types.

```
eq(a: A, a: A) → bool { … }
```

Is this loose notion of equality sufficient for path semantics? Alternatively, could the natural notion of equality mean a bit-for-bit comparison of the raw binary form?

To study this problem, I introduce two functions, one arbitrary function for encoding a binary representation of an object, and one for equality checking using some arbitrary function of the binary forms:

```
encode(a: A) → [bool] { … }

bin_eq(a: [bool], b: [bool]) → bool { … }

eq[encode × encode → id] <=> bin_eq
```

The last rule states that natural equality can be defined in terms of `bin_eq`.

The binary encoding of the object must be loss-less, which means there exists an algorithm `decode` such that:

```
encode(decode(encode(a))) = encode(a)

encode[unit → len] = \(()) = K

(`=`) = \(a: [bool], b: [bool]) → bool { all i { a[i] == b[i] } }
```

The `K` is a constant such that every bit of the objects can be compared according to this specific equality algorithm (`=`) that guarantees that `encode` is loss-less. This should not be confused with `bin_eq`, which is user defined.

To prove that you can always construct `bin_eq` from any `eq` in a particular way:

```
bin_eq[decode × decode → id] <=> eq     // Assume this kind of construction.

eq[encode × encode → id] <=> bin_eq
eq[encode × encode → id][decode × decode → id] <=> bin_eq[decode × decode → id]
eq[encode × encode → id][decode × decode → id] <=> eq
eq[(decode · encode) × (decode · encode) → (id · id)] <=> eq
eq[id × id → id] <=> eq
eq <=> eq
```

This means that natural equality can indeed be defined in terms of `bin_eq`. It makes things easier because additional constraints can be imposed on the binary form such that partial knowledge of `bin_eq` can be defined formally.

Assume there is a function `f` that operates on an object `x`. Whenever `f` makes a true/false decision that can influence the behavior of the result of `f` in any way, then this corresponds to reading a bit in the encoded representation of the object. The `encode` function is chosen such that this is always true.

It can be a bit hard to imagine why this makes sense for multi-argument functions. Picture an algorithm performing lots of instructions that read bits from more than one object, mixes the information, and makes decisions based on complicated expressions that takes the bits as input. Yet, it is certain that every bit from every object is preserved as input, such that whatever the function decides to do, it will do the exact same thing for another object that is encoded with same bits.

Therefore, natural equality can be defined to be the following function over only those bits that are read by any function using the following construction:

$$\text{eq}[\text{encode} \times \text{encode} \to \text{id}] \iff \backslash(a: [\text{bool}], b: [\text{bool}]) \to \text{bool} \{ \text{all } i \{ a[i] == b[i] \} \}$$

This means that if two objects A and B are treated as equal by any function, then they have the exact same binary encoding. Of course, a such encoding is rarely defined, but it helps to formally define the logically equivalent function that is natural, given any set of functions. *The arbitrary defined equality operator must be logically equivalent to the natural equality, or in practice limit constructed data to the range that intersects with natural equality.*

By this theorem, natural equality has a precise definition, with only one corresponding logical equivalent set of functions.

General recursive functions and arbitrary encodings can lead to an infinite number of bits in the encoded format. This means that if all functions halt that read bits, the `bin_eq` function can check a finite number of bits.

Since natural equality is commutative, it follows that the equality operator must also be commutative.