Single-Bit Languages

by Sven Nilsen, 2020

In this paper I describe the basic theory of single-bit languages.

A single-bit language is the simplest possible pure mathematical language that can be constructed. These languages only consists of two symbols, \`0\` and \`1\`. Hence, the entire language can be described as a bit. Counter-intuitively, it is not allowed to use more than one bit within the language.

For any thought, a single-bit language can be constructed that represents the thought precisely.

For example, the thought "it is raining today" can be represented as \`1\` and \`0\` is used to represent something else. When I say \`1\`, you know that I mean "it is raining today".

Another example: The unit interval \`[0, 1]\` of real numbers, can be represented in a single-bit language as \`1\`, while \`0\` is used to represent something else. If I use \`0\` to represent all real numbers, then this single-bit language can not be used to think about anything, except a single-dimensional semantics of real numbers. However, if \`0\` is used to represent anything except the exact same thing as \`1\`, then the single-bit language can be used to talk about anything. The problem is that there is not much to say.

Single-bit languages are interesting because they point out the biases that human brains have about how languages should work. Since there is no simpler pure mathematical languages, one might think that single-bit languages are over-simplifications, or in some sense "lying" about mathematical truth. However, when you actually try to prove this mathematically, it turns out that it is your own intuition about languages that is wrong. Single-bit languages are internally consistent and makes sense.

The genius thing about single-bit languages is that they contain just enough complexity to avoid lies. If you could prove that single-bit languages in general are biased, then you would also be able to prove that they are "lying" about mathematical truth. However, since one can use one symbol to represent something concretely, there is one symbol left to "fill out" the gaps of semantics, or meaning. This is enough complexity to avoid bias. Hence, single-bit languages actually makes sense.

Another reason single-bit languages are interesting is because they can be used as a tool to explore mathematical semantics. Since there is no internal structure in the language to focus on, because there is just \`0\` and \`1\`, the human brain focuses on the external structure instead: How these symbols came to mean something in the first place.

Intuitively, the only reason \`0\` and \`1\` can mean something, is because there exists an external world to the single-bit language. With other words, \`0\` and \`1\` are associated with something. Without anything to associate with them, the only thing \`0\` and \`1\` could mean were themselves. They would just be symbols with no extra semantics. There would be no world containing people to use the symbols.

Traditionally, the exploration of the boundary between the inside of a language and the outside has been associated with mysticism. In modern mathematics however, this boundary is taken for granted. There is nothing intrinsically mysterious about it. The mystery is just the feeling of not understanding entirely how it works and how to start reasoning about it.

The first lesson about boundary reasoning with single-bit languages is that one must be patient. Obviously, since single-bit languages are very different than all other languages one uses in everyday life, it takes a while to develop a deep intuition for how they work. One can not expect the same kind of progress that is done in other areas of research, because instead of trying to achieve some yet unknown result in the future, the end result is already known and seemingly trivial. The hard work is reconstructing what happens in the thought processes that can construct single-bit languages with such ease. The human brain wants to hurry, not pausing to think about stuff it erases from memory to make room for something else. Patience is a key to new insights.

One can compare the exploration process of this boundary as the activity of fishing with a rod. Instead of chasing ideas around and jumping to conclusions, one must put out a bait for the "fish" and let it come by itself in its own time. Remove distractions and silence your mind.

In path semantics, the notion of a "path" is a transformation of some function into another. When `1` is presented in a single-bit language, it means there exists a such transformation from a thought, an idea, or a source of information. The existence of this transformation gives meaning to the symbol. Hence, it is the focus on the transformation that is important for reasoning about the boundary. In general, the choice of transformation and choice of single-bit language is not relevant. What holds for all boundaries of single-bit languages also holds for all choices of transformations. When desired, one can produce examples. The simpler examples there are, the easier it is to check our intuition.

What one does when developing intuition is to build a language for expressing ideas. This language is more abstract than the specific choice of a single-bit language. It is kind of like developing a type system for thoughts. Ideas are expressed as sentences in this type system.

First, one can start with defining symbols that represents various aspects of the theory. An obvious candidate is a symbol that represents a single-bit language itself:

```
single_bit_language
```

This is a type:

```
single_bit_language : type
```

If `f` is a single-bit language, then one can model it as a function that takes some input and returns `true` or `false`.

```
f : single_bit_language
```

```
f : T → bool
```

Therefore, in this model, a single-bit language is a function type returning `bool`:

```
single_bit_language <=> T → bool
```

There are two possible thoughts in a such language:

```
x : [f] true
y : [f] false
```

So, thoughts in a single-bit language can be expressed as a sub-type.

One can immediately see that path semantics is a proper language to talk about the theory of single-bit languages. It follows from expressing ideas in the form:

> x : [f] true
> y : [f] false

That any arbitrary sub-type of the form:

> z : [g] a

Constructs a single-bit language when this judgement is taken as a truth value.
The sentence `z : [g] a` is either true or false and naturally is interpreted as intentionally true.

Therefore, when assuming the ability to express ideas in the language of path semantics:

> arbitrary sub-types ~= single-bit languages

Here, `~=` represents an isomorphism.

Previously, I showed that single-bit languages are without bias. This means that arbitrary sub-types are also without bias when used as a language.

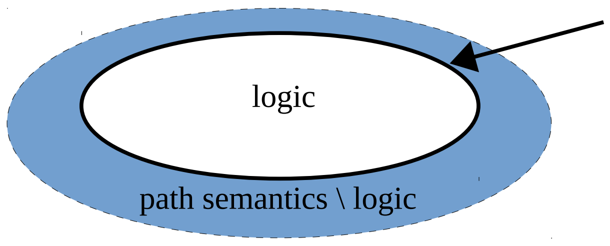
So far, we have not proven how single-bit languages come to mean something in the first place. As a theory in itself, it seems somewhat mysterious. However, when considered as a pattern hidden inside path semantics, one can justify the existence of single-bit languages as a family of normal paths.

Yet, path semantics is not the only way to model single-bit languages. One can also use logic and model single-bit languages as propositions. While it might seem like the same thing, the difference is that when you point out arbitrary sub-types as a model of single-bit languages, this is an intrinsic aspect of path semantics, while propositions in logic requires interpretation. To be more precise, the isomorphism between single-bit languages and logic is not to propositions themselves, but to the interpretation of propositions:

> interpretation of propositions ~= single-bit languages

In path semantics, expressions are considered “views” into a more complex world that can not be fully expressed at once. In logic, the interpretation is kept separated and the theory itself is manipulated purely through syntactic rules or some algorithm that grounds truth. With other words, one can say that the theory is “inside” in path semantics and “outside” in logic. This gives the boundary definition:

modern mathematics do not treat semantics of single-bit languages as some “mysterious” aspect of existence, like done in some traditional schools of thought



single-bit languages sits at the internal boundary obtained by excluding logic from path semantics and is therefore a rigorous mathematical object of study

Q.E.D.