Bit Sequence Equality Using Path Semantical Qubit

by Sven Nilsen, 2023

In this paper I discuss how one can use Path Semantical Quantum Propositional Logic (PSQ) to check two bit sequences for equality without needing infinite amount of work. When two bit sequences are encoded in PSQ in a specific way, they can be checked for equality by proving tautological propositional equality of the two first bits. This proof in turn implies the equality of pairwise bits for the rest of the sequences. I also discuss some examples of similar phenomena.

The constructive model of Path Semantical Quantum Propositional Logic (PSQ) is an extended Intuitionistic Propositional Logic (IPL)[1] with a path semantical qubit operator `~`[2].

The particular details for PSQ are highly technical and for people who are not experts in Path Semantics[3], they are irrelevant. The important thing here is to communicate the ideas and perhaps some of the intuition that goes into thinking about PSQ.

Assume that Alice has a job to compare two bit sequences which she receives from Bob. She ought to report back whether the two sequences are equal or differ by at least one bit. Bob might send two infinite bit sequences and there is no way for Alice to tell whether she received an infinite bit sequence or not. At first, this seems like an impossible job, and it is for some cases, but nevertheless Alice can make her life easier in some ways. Alice is allowed to ask Bob to send the two bit sequences in any preferred format of choice. Bob has to comply, as long the format does not reveal whether the sequences are finite or not.

This example translates into PSQ the following way:
When the first two bits in the sequence are `a` and `b`,
the next two bits are `~a` and `~b` respectively,
followed by `~~a` and `~~b` etc.

There is no way to tell in PSQ whether two such sequences terminate. Alice can use PSQ to prove whether two bits are equal, e.g. `~a == ~b`. If she finds a proof that the two bits are not equal, then she reports back to Bob that the two bit sequences are not the same. Bob encodes his knowledge of the bit sequence as propositions in PSQ, which is transferred to Alice as code. If Alice finds one particular piece hard to understand, she can ask Bob to transform the code, if possible, to some familiar structure for Alice. Alice wants to make her life as easy as possible, so she exploits Bob to maximum degree and this is where the technique to check the sequences for equality comes from.

There is a way for Alice to perform as little work as possible in a particular case.
She simply asks Bob to prove `(a == b)^true`, if possible.

The trick is that if `(a == b)^true` can be proved, then the rest of the bit sequences are equal.
This is due to a theorem in PSQ:

```
(a == b)^true   =>      (~a == ~b)^true         for any `a, b`
```

Now, there is case where Bob can prevent this exploit, while keeping the sequences equal, which is simply assuming `a == b`, `~a == ~b` and `~~a == ~~b` etc.

One can think about this as a zero-sum game, where Alice tries to minimize the work she needs to do and Bob tries to maximize the work that Alice does, but without continuing playing the game forever. However, they both need to play according to the rules.

When Alice wins, she knows `(a == b)^true` and can go home for the day.
If there is any possibility for Alice to win, then she will know by asking Bob to prove it.

Bob on the other hand, must stay on the job as long he wants Alice to stay working.
Despite being able to send the source code and go home the moment Alice has no further questions, Bob can not avoid the exploit Alice can use in a such way that he can go home while Alice remains.

Imagine that Alice first asks Bob to prove `(a == b)^true`. If Bob can not prove this, then Alice asks Bob to prove `a == b`. If Bob can not prove this, then Alice can go home. If Bob can prove `a == b`, then Alice asks Bob to prove `(~a == ~b)^true`. This repeats over and over for every bit.

There is no way for Bob to exploit Alice, despite the fact that the bit sequences might be infinite in length. Alice never knows where the bit sequences terminate, but counter-intuitively, she does not need to: The only thing Alice needs to know in order to determine that the bit sequences are equal, is to ask Bob a simple question. Bob might take long time to answer it, but he can not go home while Alice waits for the answer. Even in the case where Bob has a non-trivial relationship in mind for two bits, Alice can ask Bob a simple question to make progress. This also requires Bob to stay working while Alice works. The only way for Bob to go home is to tell two bits are unequal or that the rest of the bits are equal. In every case, Alice wins.

Another version of this thought experiment is when Cinderella[4] wants to go to the ball to see the prince, her evil step-mother tries to keep Cinderella home, busied by work. However, the paradox is that the evil step-mother is not able to go to the ball herself. As much she wants to keep Cinderella enslaved, she also wants to go the ball and the moment she goes, so will Cinderella.

The metaphor is that in some cases, it is not possible to outsmart someone that one in principle can exploit for labour. The labour exploit is not impossible, but the price is too high for any rational agent to engage in it as an activity. This is also a metaphor for the human condition, where labour exploiters often end up working themselves to death, in attempt to control the workers they exploit, no matter how intuitive it seems that the labour exploit is working. As counter-intuitive as it sounds, the usual outcome is that labour exploiters need to work almost as much as the workers being exploited. The significant difference is in health hazards, payment and one-to-many relationships.

References:

[1] “Intuitionistic logic”
Wikipedia
https://en.wikipedia.org/wiki/Intuitionistic_logic

[2] “Path Semantical Qubit”
Sven Nilsen, 2022
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip2/path-semantical-qubit.pdf

[3] “Path Semantics”
AdvancedResearch
https://github.com/advancedresearch/path_semantics

[4] “Cinderella”
Wikipedia
https://en.wikipedia.org/wiki/Cinderella