A New Foundation for Quantum Programming

David Wakeham
david@ torsor.io
Founder, Torsor

Overview

Qubits are a nice way to build a quantum computer but a terrible way to program one. We propose a new "operator first" abstraction layer in which operators are primitive and states derived. We explain how it works mathematically, how it can be used to optimize circuits, how to code without qubits and how to compile to them.

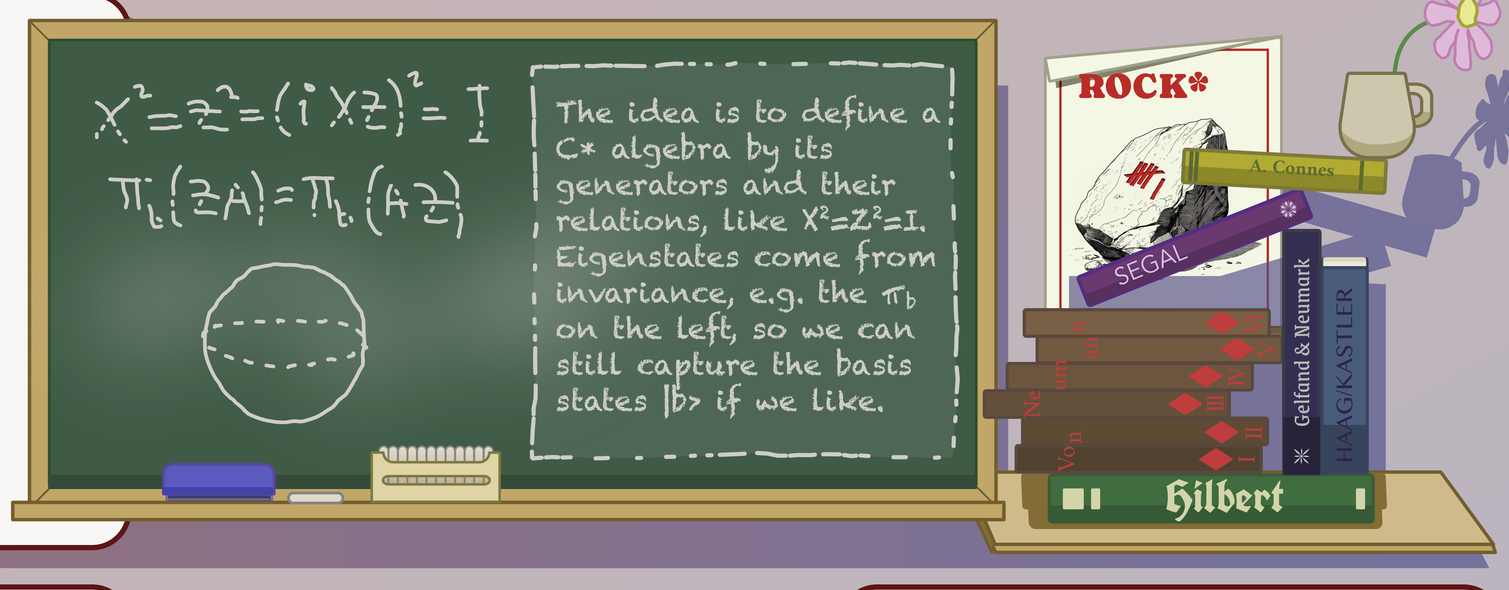
Our software stack will consist of:

- a rigorous mathematical foundation;
- a new circuit calculus for operators;
- a DSL⚠ for building operator circuits;
- a backend-agnostic compiler.⚠

Abstractions are only as good as the things they can do. Beyond simplifying software and circuits, our operator layer should enable other fun things: combining CV and DV, integrated error correction, a simpler interface to open system dynamics, and new types of hybrid workflow (with applications to QML). Ask me anything!

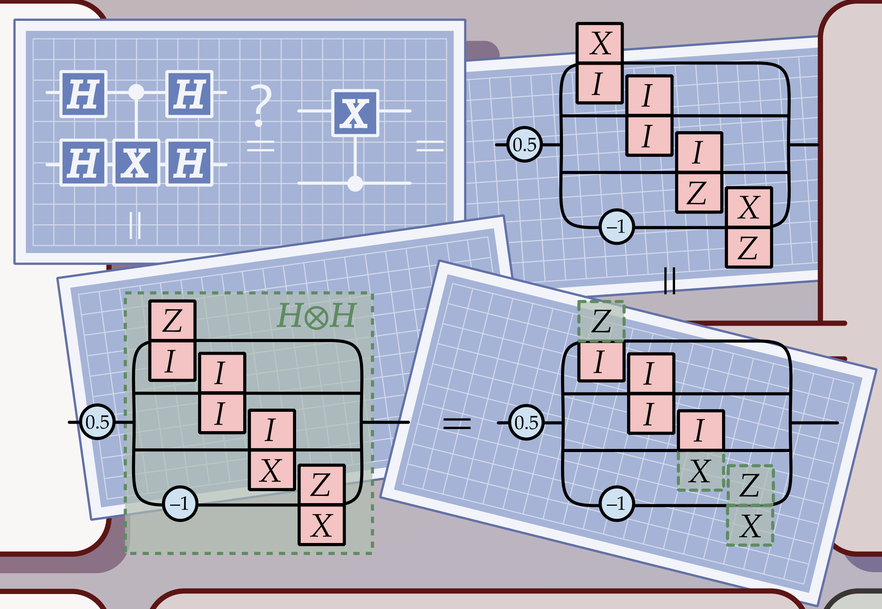
Mathematics

Hilbert space is dead; long live hilbert space. Our approach is governed by the *C* algebra*, a more flexible entity which captures the relationship between observables. A state φ can then be viewed as a *function* taking observables X to expectations $\varphi(X) = \langle X \rangle_\varphi$. Once specified, a given algebra can be "interpreted" as a set of operators on a canonical Hilbert space by the *GNS construction*.



Logic circuits

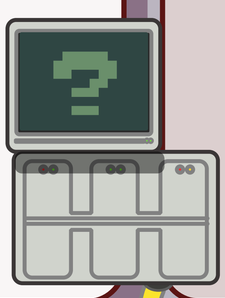
Classical computing is about algebras, not states. Shannon baked Boolean algebra into circuits and used logico-algebraic methods to optimize them. We can do the same, replacing Boolean by C* and using symbolic laws to optimize circuits. In this light, the GNS construction plays the role of truth tables. In contrast, conventional quantum circuits are based on state-oriented "conservative" logic.



Proof that conjugating a CNOT by Hadamards swaps wires. We expand sums in parallel, products in series, and tensor stacked gates. The dotted box is a channel. We use universal algebraic laws, local definitions (e.g. $2\Pi_0 = I+Z$) and local transpiles (e.g. $HZH = X$) along the way. Ask for more info!

Programming

We usually know what we want even if we don't know how to get it. In complexity theory, oracles formalize this idea; high-level languages are also "oracular" in the sense that programming constructs do not force low-level implementation details on us. We propose *algebraic relations* as the oracular foundation of a high-level DSL. This is grounded in the formalism of *universal C* algebras*.



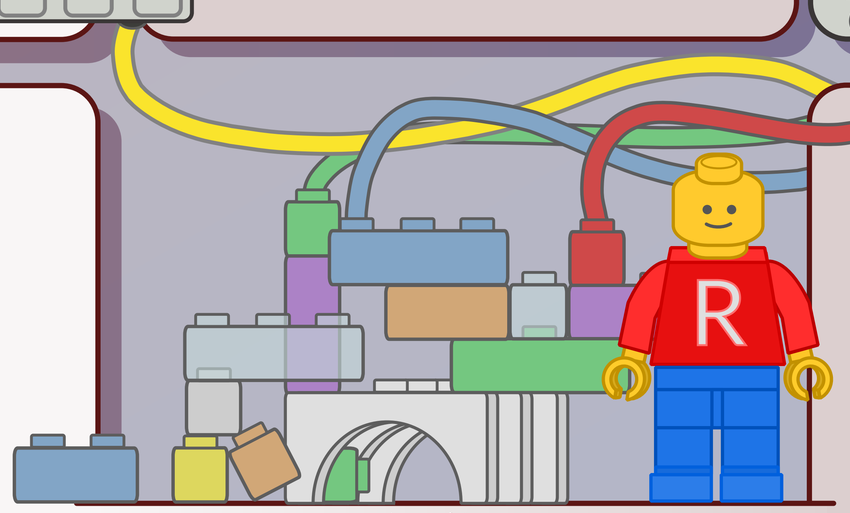
Programming with mathematical objects means that the declarative paradigm is a natural fit. We embed our DSL in **Haskell**, a powerful, elegant and typesafe functional language. On the right, the code snippet shows how to define the Pauli algebra and prepare a Bell state.

```
[x, z] = [gen 'x', gen 'z']            -- define (Hermitian) generators
y = (:i)x*z                            -- define expression from gens
pRels = [x**2-1, y**2-1, z**2-1]       -- define relations to set to 0

Pauli = Algebra                        --  define algebra from gens/rels
        { gens = [x, z]
        , rels = pRels
        }

Pauli2 = TensorAlg [Pauli, Pauli]      -- tensor product algebra

pi0 :: State Pauli
pi0 = eigen Z 1                        -- define |0> using invariance

cnot :: Pauli2
cnot = (*) 0.5 $ (z-1)#1 + (z+1)#1     -- build CNOT (# = tensor prod)

bellPair :: State Pauli2               -- define Bell pair state
bellPair = let h = (x + z) / sqrt 2    --   define Hadamard
               init = pi0 # pi0        --   define initial prod state
           in sapply cnot $            --   apply CNOT after Hadamard
              sapply (h # 1) init      --   (sapply = apply to state)
```

⚠ = work in progress

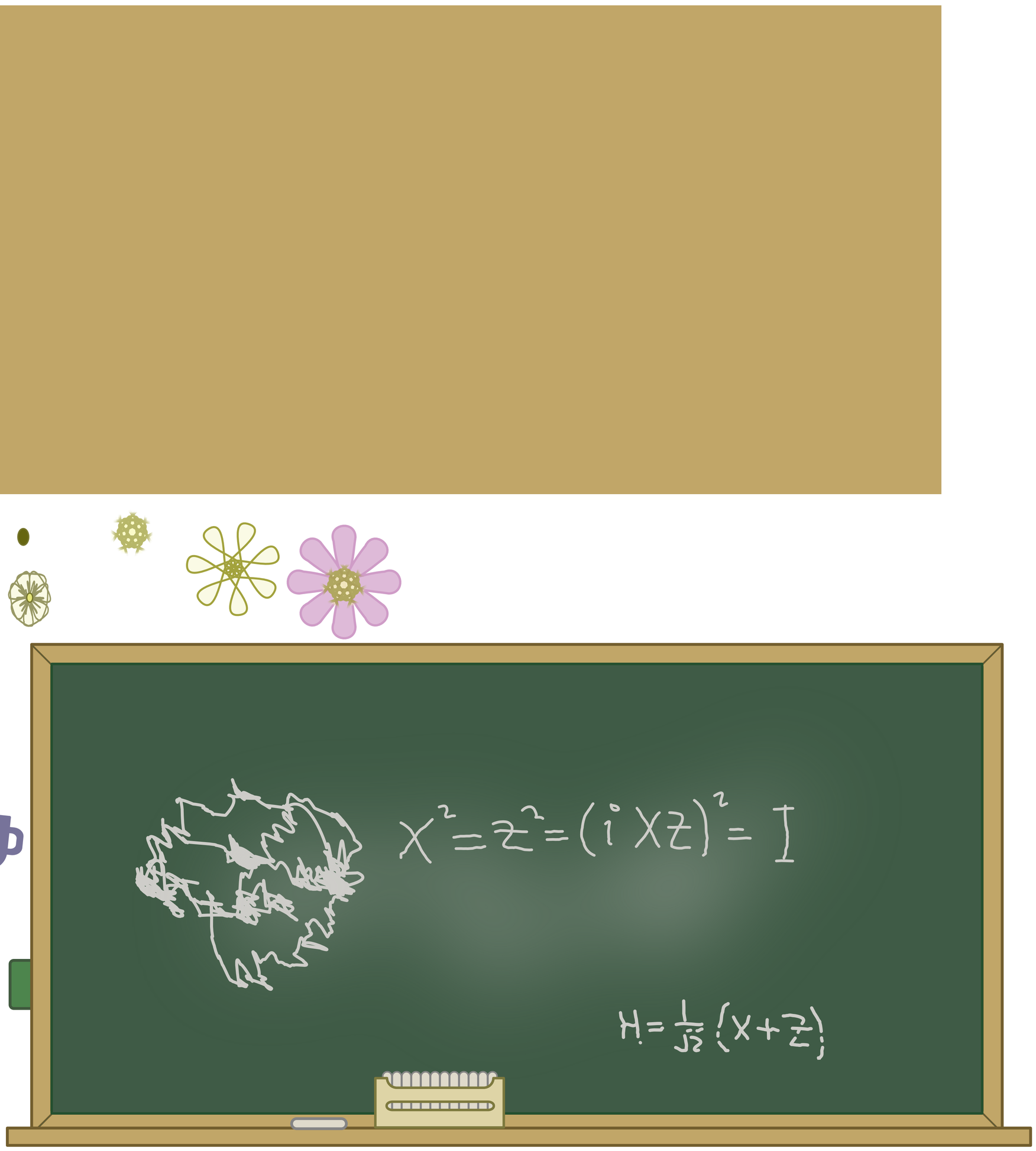
Compilation

High-level giveth and high-level taketh away. We can ask for cool things, but how do we know the low-level circuit can provide? There are three related ways to approach this: (a) *functional-analytic*, yielding a fundamental compilability resut; (b) using *error correction* to approximately embed the GNS Hilbert space in hardware; and (c) *bisimulation*, a logical/computational equivalence.



⚠ = work in progress

The tensor product of an infinite list of finite algebras is called *hyperfinite*. Remarkably, the result R is *independent of factors*! This is an algebraic form of the Church-Turing thesis, and implies we can compile a finite algebra, to any precision, using qubits, qudits, or whatever parts are lying around!


$X^2 = Z^2 = (iXZ)^2 = I$
$H = \frac{1}{\sqrt{2}}(X+Z)$