I'm curating a list of open research questions related to Anoma's PLT subcomponents. Feel free to respond with anything relevant; I'll edit the top post to keep it up-to-date.

What is the optimisation space for through-VM vs. direct circuit compilation?

When comparing circuit compilation approaches, there's a trade-off axis between VM-based execution, which adds overhead but allows proofs to be made only for the actual execution path, and direct circuit compilation, which is lower overhead but requires that proofs be made for all possible execution paths (branches). In principle, it is possible to combine these approaches such that parts of a program (sub-programs) are compiled directly to circuits, while parts are interpreted. What does the optimisation space look like for this compilation approach?

Can we cleanly characterize the program optimisation problem?

With a suitably abstract intermediate representation such asGEB, if we compile programs either directly to circuits with known cost models (costs for addition gates, multiplication gates, custom gates, depth, height, etc.) or to VMs with known cost models, can we characterise the optimisation problem cleanly enough that it can be handed off to existing NP optimisation techniques (SAT solvers, simulated annealing, learning algorithms, etc.)? How does this interact with translating between representations with different carrier fields (e.g. curves for a ZK proof system, binary for typical computers)?|

Can we use information flow control techniques to build cryptographic primitive synthesis?

Can we adapt and extend current mathematical techniques for reasoning about the information flow properties of programs, such as hyperproperties (link), to a sufficiently powerful setting (including e.g. statistical reasoning about difficulty of hidden variable inference) that they can be used for cryptographic primitive synthesis? (for example, ZK proof systems)?