Introduction
What is Cairo?
Cairo is a programming language designed for a virtual CPU of the same name. The unique aspect of this processor is that it was not created for the physical constraints of our world but for cryptographic ones, making it capable of efficiently proving the execution of any program running on it. This means that you can perform time consuming operations on a machine you don't trust, and check the result very quickly on a cheaper machine.
While Cairo 0 used to be directly compiled to CASM, the Cairo CPU assembly, Cairo 1 is a higher level language. It first compiles to Sierra, an intermediate representation of Cairo which will compile later down to a safe subset of CASM. The point of Sierra is to ensure your CASM will always be provable, even when the computation fails.
What Can you Do with It?
Cairo allows you to compute trustworthy values on untrusted machines. One major usecase is Starknet, a solution to Ethereum scaling. Ethereum is a decentralized blockchain platform that enables the creation of decentralized applications where every single interaction between a user and a d-app is verified by all the participants. Starknet is a Layer 2 built on top of Ethereum. Instead of having all the participants of the network to verify all user interactions, only one node, called the prover, executes the programs and generates proofs that the computations were done correctly. These proofs are then verified by an Ethereum smart contract, requiring significantly less computational power compared to executing the interactions themselves. This approach allows for increased throughput and reduced transaction costs while preserving Ethereum security.
What Are the Differences with Other Programming Languages?
Cairo is quite different from traditional programming languages, especially when it comes to overhead costs and its primary advantages. Your program can be executed in two different ways:
- When executed by the prover, it is similar to any other language. Because Cairo is virtualized, and because the operations were not specifically designed for maximum efficiency, this can lead to some performance overhead but it is not the most relevant part to optimize.
- When the generated proof is verified by a verifier, it is a bit different. This has to be as cheap as possible since it could potentially be verified on many very small machines. Fortunately verifying is faster than computing and Cairo has some unique advantages to improve it even more. A notable one is non-determinism. This is a topic you will cover in more detail later in this book, but the idea is that you can theoretically use a different algorithm for verifying than for computing. Currently, writing custom non-deterministic code is not supported for the developers, but the standard library leverages non-determinism for improved performance. For example sorting an array in Cairo costs the same price as copying it. Because the verifier doesn't sort the array, it just checks that it is sorted, which is cheaper.
Another aspect that sets the language apart is its memory model. In Cairo, memory access is immutable, meaning that once a value is written to memory, it cannot be changed. Cairo 1 provides abstractions that help developers work with these constraints, but it does not fully simulate mutability. Therefore, developers must think

carefully about how they manage memory and data structures in their programs to optimize performance.

References

- Cairo CPU Architecture: <https://eprint.iacr.org/2021/1063>
- Cairo, Sierra and Casm: <https://medium.com/nethermind-eth/under-the-hood-of-cairo-1-0-exploring-sierra-7f32808421f5>
- State of non determinism: <https://twitter.com/PapiniShahar/status/1638203716535713798>