Circuits must be fixed in size. However, one is often faced with dynamically sized objects which then require either recursion or dedicated circuits for every single size.

At ETHGlobal London last weekend we were working on circuits for proving audio processing. One of the challenges was that audio files varying in size and processing history require different circuits for the same operation. This led us to create a small templating framework for Noir to dynamically fill in parts of the circuit just before compilation once the required specifications for the circuit were known. That way we only needed one circuit template for each operation from which we could generate any circuits needed. We were able to reduce any changes or adjustments in the circuit to a set of global variables that controlled the final circuit.

github.com

LucasAschenbach/content-commitment-protocol/blob/main/editor/circuits/compress/src/main.nr.template

use dep::std;

global SIZE = <%= content_size %>; // size of the sound array global SIZE_OLD = <%= old_content_size %>; // size of the old sound array global LAST_OP_ARGS = <%= last_op_args %>; // size of the previous operation arguments

[recursive]

fn main(// Commitment ---------- com: pub Field, // Content Operation ---------- sound_new: pub [Field; SIZE], sound_old: [Field; SIZE_OLD], // op args // compression_power: Field, // Previous proof data ---------- verification_key : [Field; 114], proof : [Field; 93], last_op_args: [Field; LAST_OP_ARGS], key_hash : Field,

This file has been truncated. show original

For our use case, having something like “CLI arguments” that are passed to noir_wasm and can be used inside the circuit would have already been sufficient. What do you think about adding something like this to the Noir language compiler?