3.2 Simplifying Complex Components

Assume that we have a computing machine M, that accepts an input in and produces an output out. We want to calculate the output that results from applying M on an initial input for n times. Note that M may not be state-less:

$$M_n(in_1, out_1, out_2, \ldots, out_{n-1}) \coloneqq out_n$$

We assume that M is composed of a number of subcomponents. Every subcomponent is like a computing machine and accepts an input and produces an output. Like M, subcomponents are not necessarily state-less. When M receives an input, based on the input, a number of subcomponents gets activated, they generate their output from the input, and then their outputs are combined to produce the output of M.

To build a verifying circuit for this computation, we need to construct a circuit for M and then repeat this circuit n times to compute out_n at the final step of the computation.[1]

Assume that we have a subcomponent that has a considerable arithmetic circuit complexity. Even if this subcomponent is active in only a few computation steps, We will still have to repeat its circuit in every step of the computation. This will increase the complexity of proof generation considerably. Fortunately, there is a workaround. Instead of repeating the circuit of the component, we can use a simpler cryptographic hash calculator circuit during the computation. At the end, to verify the functionality of the component, we use a final verification circuit.

Every subcomponent accepts an input in and produces an output out. If the component gets activated for k steps, we can denote it by a deterministic function that maps an input sequence with length k, to an output sequence with the same length:[2]

$$f((in_1, in_2, \ldots, in_k)) \coloneqq (out_1, out_2, \ldots, out_k)$$

In every step of the computation, we replace this component with a cryptographic hash calculator which receives in and out as its inputs and gets activated in the same computation steps that the component must be active. When the hash calculator is active, it hashes its inputs, so at the end of the computation it has computed a digest:

$$h(in_1, out_1, in_2, out_2, \ldots, in_k, out_k) \coloneqq digest_f$$

where k is the number of steps that our component must have been active.

Now the prover needs to prove that he knows values $in_1, out_1, \ldots, in_k, out_k$ such that they are correctly produced by the component:

$$f((in_1, \ldots, in_k)) = (out_1, \ldots, out_k)$$

[1] Since M is not state-less, we may have to feed inputs from steps, say, $i-1, i-2, \ldots$ to step i.

[2] Note that subcomponents are not active in every step of the computation, so in_i is **not** indicating the input of the component at the ith step of the computation.

and their digest is also correct:

$$h(in_1, out_1, \ldots, in_k, out_k) = digest_f$$

where functions f and h, are both known to the prover and verifier.

The circuit for verifying this assessment usually is straight forward. When the computation involves a large number of steps and the component is complex, this approach can reduce the cost of proof generation considerably. Memory components are good candidates for being simplified by this method.

Interestingly, this approach can also reduce the number of computation steps. When we use a hash calculator circuit instead of our component, *out* will be available in the same computation step that *in* is available. This will eliminate one computation step that is needed for generating *out* by the component's circuit.