This follows up on my earlier postgenSTARK: a JavaScript zk-STARK generation framework.

I've implemented a first prototype of a simple language for writing AIR constraints for zk-STARKs. The language is called AirScript. The new version (v0.4) of genSTARK library, which I just released, now relies on AirScript for STARK definitions.

Here is an example of how a MiMC STARK can be defined using AirScript:

```
define MiMC over prime field (2^256 - 351 * 2^32 + 1) {

transition 1 register in 2^13 steps {
    out: $r0^3 + $k0;
}

enforce 1 constraint of degree 3 {
    out: $n0 - ($r0^3 + $k0);
}

using 1 readonly register {
    $k0: repeat [...]; // actual 64 constants go between the brackets
}

}
```

And here is AirScript for a modified version ofRescue hash function:

```
define Rescue over prime field (2^64 - 21 * 2^30 + 1) {

alpha: 3;
inv_alpha: 0-6148914683720324437;

MDS: [
    [18446744051160973310, 18446744051160973301],
    [                   4,                   13]
];

INV_MDS: [
    [ 2049638227906774814,  6148914683720324439],
    [16397105823254198500, 12297829367440648875]
];

transition 2 registers in 32 steps {
    S: [$r0, $r1];
    K1: [$k0, $k1];
    K2: [$k2, $k3];
    S: MDS # S^alpha + K1;
    out: MDS # S^(inv_alpha) + K2;
}

enforce 2 constraints of degree 3 {
    S: [$r0, $r1];
    N: [$n0, $n1];
    K1: [$k0, $k1];
    K2: [$k2, $k3];

    T1: MDS # S^alpha + K1;
    T2: (INV_MDS # (N - K2))^alpha;

    out: T1 - T2;
}

using 4 readonly registers {
    $k0: repeat [...]; // actual 32 constants go between the brackets
    $k1: repeat [...]; // actual 32 constants go between the brackets
    $k2: repeat [...]; // actual 32 constants go between the brackets
    $k3: repeat [...]; // actual 32 constants go between the brackets
}

}
```

You can see complete exmaples of these STARKshere.

Input injection

v0.4 of genSTARK library also supportsInput injection. This basically allows aggregating proofs of the same computation for different inputs into a single proof.

For example, we could aggregate proofs of knowledge of Rescue hash preimage for 16 values into a single proof. The resulting proof is ~114 KB in size (while a proof for a single value is ~37 KB in size). You can see more benchmarks here.

Future plans

AirScript is not yet expressive enough to support easy definitions of more complex STARKs. For example, defining a STARK that could prove membership of a value in a Merkle tree is rather cumbersome. This is something I’m planning to address next.

If you have any thoughts or feedback on these, let me know!