R1CS Programming

ZK0x04 Workshop Notes

Daniel Lubarov   Brendan Farmer

October 16, 2019

Contents

1 Multiplicative inverse

Deterministically computing $1/x$ in an R1CS circuit would be expensive. Instead, we can have the prover compute $1/x$ outside of the circuit and supply the result as a witness element, which we will call x_{inv}. To verify the result, we enforce

$$(x)(x_{\text{inv}}) = (1) \tag{1}$$

2 Zero testing

To assert $x = 0$, we simply enforce

$$(x)(1) = (0) \tag{2}$$

Asserting $x \neq 0$ is similarly easy: we compute $1/x$ (non-deterministically, as in Section 1). The result can be ignored; the mere fact that an inverse exists implies $x \neq 0$.

On the other hand, if we want to *evaluate*

$$y = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

we can do so by introducing another variable m, and enforcing

$$(x)(m) = (y) \tag{4}$$
$$(1 - y)(x) = (0) \tag{5}$$

TODO: Explain further. This method is from [1].

3 Binary

To assert $b \in \{0, 1\}$, we enforce

$$(b)(b - 1) = (0) \tag{6}$$

TODO: Discuss splits and joins

4 Comparisons

TODO: Describe basic comparison algorithm

TODO: Describe Ahmed's optimization

A few other optimizations are possible in particular circumstances:

1. To assert (not evaluate) $x < y$, we can split x non-canonically and split y canonically. The prover is forced to use x's canonical representation anyway, otherwise $x_{\text{bin}} \geq |F| > y_{\text{bin}}$, making the assertion unsatisfiable.
2. To assert $x < c$ for some constant $c \ll |F|$, we can split x into just $\lceil \log_2(c) \rceil$ bits.

5 Permutations

Say we want to verify that two sequences, $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$, are permutations of one another.

6 Sorting

TODO: Discuss sorting networks

TODO: Discuss permutation networks + comparisons to verify order

7 Random access

TODO: Discuss naive random access via index comparisons

TODO: Discuss binary tree method

8 Embedded curve operations

TODO: Discuss basic embedded curve operations

References

[1] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*, pp. 238–252, IEEE, 2013.