

R1CS Programming

ZK0x04 Workshop Notes

Daniel Lubarov Brendan Farmer

October 16, 2019

Contents

1	Multiplicative inverse	2
2	Zero testing	2
3	Binary	2
4	Selection	3
5	2x2 switch	3
6	Permutations	3
6.1	2x2 case	3
6.2	2x2 case	3
6.3	2x2 case	3
7	Sorting	3
8	Random access	4
9	Comparisons	4
10	Embedded curve operations	4

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

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

2 Zero testing

To assert $x = 0$, we simply enforce

$$(x)(1) = (0) \quad (2)$$

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*

$$y := \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

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

$$(x)(m) = (y) \quad (4)$$

$$(1 - y)(x) = (0) \quad (5)$$

Outside of the circuit, the prover generates y as in [Equation 3](#), and generates m as

$$m := \begin{cases} 1 & \text{if } x = 0 \\ y/x & \text{otherwise} \end{cases} \quad (6)$$

This method is from [\[1\]](#).

3 Binary

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

$$(b)(b - 1) = (0) \quad (7)$$

TODO: Discuss splits and joins

4 Selection

Suppose we have a boolean value s , and we wish to compute

$$z := \begin{cases} x & \text{if } s = 0 \\ y & \text{if } s = 1 \end{cases} \quad (8)$$

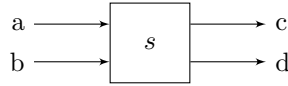
We can compute this as

$$z := x + s(y - x) \quad (9)$$

This requires two constraints: one “is boolean” assertion (Equation 7), assuming s was not already known to be boolean, and one for the multiplication.

5 2x2 switch

Suppose we wish to implement a switch with the following structure:



In particular, if $s = 0$ then the outputs should be identical the inputs: $(c, d) = (a, b)$. If $s = 1$ then the inputs should be swapped: $(c, d) = (b, a)$.

This requires two constraints: one “is boolean” assertion (Equation 7), and another for selecting the value of c (Equation 9). Once we have c , we can compute d “for free” as

$$d := a + b - c \quad (10)$$

6 Permutations

Say we want to verify that two sequences, (x_1, \dots, x_n) and (y_1, \dots, y_n) , are permutations of one another. This can be done efficiently using routing networks, which used a fixed (for a fixed n) network of 2x2 switches.

AS-Waksman networks [2] are a particularly useful construction, since they support arbitrary permutation sizes. They use about $n \log_2(n) - n$ switches, which is close to the theoretical lower bound of $\log_2(n!)$.

7 Sorting

TODO: Discuss sorting networks

TODO: Discuss permutation networks + comparisons to verify order

8 Random access

TODO: Discuss naive random access via index comparisons

TODO: Discuss binary tree method

9 Comparisons

TODO: Describe basic comparison algorithm

TODO: Describe Ahmed’s optimization

A couple 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.

10 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.
- [2] B. Beauquier and E. Darrot, “On arbitrary size waksman networks and their vulnerability,” *Parallel Processing Letters*, vol. 12, no. 03n04, pp. 287–296, 2002.