J-R1CS – a JSON Lines format for R1CS

Guillaume Drevon
gd@sikoba.com

Sikoba Research *

February 2019

Abstract

R1CS (rank-1 constraint systems) define a set of bi-linear equations which serve as constraints suitable for ZK proofs. They describe the execution of statements written in higher-level programming languages and are used by many ZK proof applications, but there is as yet no standard way of representing them. In this paper, we present J-R1CS, a simple and lightweight *JSON Lines* format dedicated to R1CS.

This document was written in response to the call for Community Standards of the 2nd ZKProof Standards Workshop, under the Implementation topic.

1 Background and Motivation

R1CS is defined in [BCG+13] as a simpler reformulation of QAP (Quadratic Arithmetic Program) that were used previously. R1CS has de-facto become the low-level language for representing statements to be processed by ZK proof systems.

The zkproof community has started to work on a file format for R1CS in the implementation track of the first workshop [Imp]. However this binary file format is not practical for prototyping and quick developments.

Using the definitions introduced in the above mentioned workshop, we propose a *JSON Lines* format which has advantages in term of usability, readability and interopability, making it a good choice for pre-production usages. Indeed, JSON parsers/deserializers exist in many programming languages and JSON has become a widely used format for describing data for both humans and machines, due to its simplicity and compactness.

Note that the J-R1CS format is not finalized and there are still open questions.

*Supported by Fantom Foundation

2 Model

R1CS can be easily described with a *JSON Lines* format [Jso]. Since a R1CS can contain a very large number of equations, it would not be practical to have all them included in one JSON object because then the parsing would load everything in memory at once – this is the drawback of a format proposed in [whi]. Of course one could use a specific parser, but that would defeat the advantages of using JSON.

The first line is a header describing the properties of the system. Listing 1 provides an an example of the R1CS header in JSON format. For readability it is shown with line breaks, but in an J-R1CS file it should be all in one line.

```
{
   "r1cs":{
      "version":"1.0",
      "field_characteristic":"133581199851807797997178235848527563401",
      "extension_degree":1,
      "instances":3,
      "witnesses":5,
      "constraints":2000,
      "optimized":true
   }
}
```

Listing 1: JSON R1CS Header

Then we simply list every R1C line by line. As before, line breaks in Listing 2 are for readability only, the actual R1C line should not have any line breaks.

```
{
   {
      "A": [[0,"2"],[-1,"6"],[5,"4"],....],
      "B": [[0,"5"],[-6,"3"],[3,"4"],....],
      "C": [[0,"3"],[-1,"2"],[-2,"7"],[3,"4"],[4,"1"]...]
   }
}
```

Listing 2: JSON R1C

The corresponding R1C is $\langle A, X\rangle \cdot \langle B, X\rangle = \langle C, X\rangle$, where A, B and C are vectors of coefficients and X is the vector of variables (constant, instances and witnesses). In the JSON object, instead of putting all the coefficients for A,B and C, we add their index which allows to omit zero coefficients in the representation. Indeed in practice, there will be many variables in the system but only a few within one constraint. We follow the convention in [Imp] for the indexes; zero represents a constant, a negative index indicates an instance variable and positive index, a witness variable.

$$\begin{cases} index = 0 & \text{constant input} \\ index < 0 & \text{index of the instance inputs} \\ index > 0 & \text{index of the witness inputs} \end{cases} \tag{1}$$

3 Limitations and Improvements

3.1 Big integers

Although in theory JSON does not limit numbers, in practice number size will be limited by the JSON parser implementations, see [rfc] for more information. This is why we are using strings for the coefficients which will need to be handled as big integers in the ZK proof application itself.

3.2 Order of R1C vectors

By default, J-R1CS does not require the elements of the R1C vectors to be ordered by index, as suggested in [Imp]. However, ordering these elements is definitely a good practice so it can be enforced by setting the "optimized" property to true.

3.3 Variable names

In the future, there may be DSLs which will allow to identify variables by name. In some situations, it might be useful to easily identify where in the R1CS these variables appear. We could therefore add the (optional) possibility of adding names for variables.

```
{
   "names": [[-1,"IN_A"],[-2,"IN_B"],.... [1,"W_A"],[2,"W_B"],.... ],
}
```

Listing 3: Optional variables names

4 Definitions

R1CS Rank 1 Constraint Systems. It is a NP-complete language for specifying relations, as a system of bilinear constraints.

version The version number of the R1CS JSON format. It is a string, its typical format will be "Major.minor".

field_characteristic The characteristic of the underlying field in which the coefficients belong to.

extension_degree Optional property, it is the degree of the field extension. Current ZK proof systems all use 1. Default value is 1.

instances The number of instance variables, corresponding to the public inputs.

witnesses The number of witness variables, corresponding to the private inputs.

constraints The number of constraints (of rank 1).

optimized Optional property which indicates if the vectors of the constraints are ordered by index and contain no 0 coefficients. The default value is "false". When set to "true", all indices inside a vector must be unique and ordered, and the coefficients cannot be 0. The ordering must be as follows: first index 0, then the negative indices (in decreasing order: -1, -2, ...), finally the positive indices (in increasing order).

References

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[Imp] Implementation track. `https://zkproof.org/documents.html`. zkproof-implementation-20180801.pdf.

[Jso] Json lines. `http://jsonlines.org/`.

[rfc] Rfc7159. `https://tools.ietf.org/html/rfc7159#section-6`.

[whi] libsnark-tutorial. `https://github.com/barryWhiteHat/libsnark-tutorial/blob/master/zksnark_element/r1cs.json`.