S polynomial transformation for permutation argument

Decomposition

Following the original suggestion $s(X,Y) = X^{-N-1}Y^N s_1(X,Y) - X^N s2(X,Y)$

$$s_1(X,Y) = \sum_{i=1}^{N} u_i'(Y)X^{-i+N+1} + \sum_{i=1}^{N} v_i'(Y)X^{i+N+1} + \sum_{i=1}^{N} w_i'(Y)X^{i+2N+1}$$

$s_2(X,Y)$ is not important for this discussion. $s_1(X,Y)$ is in total a polynomial of degree $3N+1$.

$$u_i'(Y) = \sum_{q=1}^{Q} Y^q u(q,i)$$

and with a similar form for $v'(Y)$ and $w'(Y)$

$u(q,i)$ by itself is a constant in q-th linear constraint in front of a variable $a(i)$. $v(q,i)$ and $w(q,i)$ have the same meaning for $b(i)$ and $c(i)$.

In total $s_1(X,Y)$ can be represented as a large convolution in a form $M_{q,i}N^qK^i$ where summing is over the same index that is placed up and down. Vectors are $N^q = [Y, Y^1, ..., Y^Q]$ and $K^i = [X, X^2, ..., X^{3N+1}]$, so the matrix $M_{q,i}$ is sparse and q-th row is formed by the concatenation of coefficients of $u(q,i)$, $v(q,i)$ and $w(q,i)$ (i notation is abused). For two multiplication gates (giving variables $a(1), a(2), ..., c(2)$) and a linear constraint $10a(1) - b(1) - c(2) = 0$ a first row would look like

$$[10, 0, -1, 0, 0, -1]$$

There are three questions:

- Original paper states that $s_1(X,Y)$ can be represented as a sum of three polynomials, each of those being a permutation by itself. Why three? One could try to transform a whole matrix $M_{q,i}$ to have one permutation argument.
- If s_1 is split into sum of three polynomials, are those polynomials each form an individual permutation argument for components like $\sum_{i=1}^{N} u_i'(Y)X^{-i+N+1}$?
- What would be the most efficient procedure to do such a reduction? Just from an example above with a single constraint in a form $[10, 0, -1, 0, 0, -1]$ a first element will contribute in a summand $10X^2Y^1$, while to make a permutation argument one has to first create a term $10X^2Y^2$.

Continuing discussion

Implementation of a permutation argument requires to have some diagonal matrix $D_{q,i}$ to first commit to the combination like $\sum_{i=1}^{N} d_i X^i Y^i$ and later make a permutation argument to prove evaluation of

$\sum_{i=1}^{N} d_{\sigma(i)} X^i Y^{\sigma(i)}$ for a fixed permutation $\sigma(i)$.

In principle such requirement means that decomposition of our $M_{q,i}$ matrix into the sum of j matrixes (let's call them $J^j_{q,i}$ should have only a single coefficient in every row, so one can define a proper diagonal $D^j_{q,i}$. Such decomposition and reduction needs to be done only once per circuit, cause $D^j_{q,i}$ and corresponding $\sigma^j(i)$ will become fixed as a part of the specialized common reference.

One can not directly guess how many linear constraints and multiplication gates will be in a system. For example, trivial (w/o optimizing run, as given in the original SONICs implementation) reduction of R1CS will have number of multiplication gates equal to the $m/2 + n$ where m is a number of variables and n is a number of constraints in R1CS.

Let's take an assumption that $N > Q$, so a final constraint system will have more multiplication gates that linear constraints. In this case one can propose the following reduction procedure:

- Forbid constraints that have a form $A(1) + A(1) + \cdots$, basically require a deduplication step.
- Each constraints may have one variable of flavors A, B and C to ensure that $M_{q,i}$ has only one coefficient for variable of each kind. In this case it can already be decomposed at three matrixes $J^j_{q,i}$.
- Every constraint that breaks this rule will give raise to a new linear constraint(s) and new multiplication gate(s).
- Final constraints can NOT have zero coefficients in front of any flavor of variables, otherwise a permutation argument can not be made (it's a necessary assumption for grand product argument).
- Reduction itself is not trivial! Let's do it step by step.
- Constraint where $NumVar(A) > NumVars(B) > NumVars(C)$, so in a linear term number of contributions from variables of flavor A is greater than from flavor B, that is in term is greater than for a flavor C. In this case one can reduce $NumVar(A)$ and $NumVars(B)$ by one, and increase $NumVar(C)$ by one, through the introduction of a constraint $A + B - C = 0$. This also gives one extra multiplication gate. One can continue this procedure until there is a linear constraint in one of the following forms:
    - $A + B + C = 0$, then reduction is over
    - $A + B = 0$.
    - $A = 0$ (this in a case of public inputs also)
- $A + B = 0$ allows to try to make a constraint system in the form $A(1) + B(1) - C(2) = 0$, $A(1) + B(1) + C(2) = 0$ with those two constraints going into the different $J^j_{q,i}$.
- $A = 0$ if inflated into constraints $A(1) + B(2) - C(2) = 0$, $A(1) - B(2) + C(2) = 0$ (TODO: check the prefactors).
- To have $j = 3$ one can not allow any variable of any flavor to happen more than 3 times in all the linear constraints. Otherwise for any permutation $\sigma^j(i)$ one can not "choose" a corresponding linear constraint index.

Now this "simple" list of rules can be implemented :)