# REVIEWER 1

At least two proposals to unify relational algebra and linear algebra have been proposed in the community: LARA and MATLANG. Both languages have their shortcomings in that they can not express certain matrix operations. As matrix operations can always be added to both languages, a basic question that arises is what are the basis operations that need to be added to support a large class of matrix operations. Rather than focussing on specific matrix operations/algorithms this paper takes a more principled approach and defines an extension of MATLANG that captures a robust class of linear algebra algorithms, those defined by arithmetic circuit families. The extension of FOR-MATLANG is defined in terms of for-loops that iterate over the canonical vectors to simulate bounded iteration (say, over the dimension of a matrix). The extension is both elegant and effective (as the main technical result shows). In addition, fragments of FOR-MATLANG are distinguished corresponding to more well-known fragments of (extensions of) the relational algebra.

The present paper contains all necessary ingredients for a solid PODS paper. The problem is well-motivated (what is a core language for linear algebra), the solution is elegant (looping extension taking the characteristics of matrices into account) and effective (capture a robust class of linear algebra algorithms). Furthermore, meaningful restrictions of the language are provided as well. Strong contribution for PODS.

**(OMITED)** On a different note, for-loops where considered before in the community as an extension of the relational algebra: Frank Neven, Martin Otto, Jerzy Tyszkiewicz, Jan Van den Bussche: Adding For-Loops to First-Order Logic. Inf. Comput. 168(2): 156-186 (2001). Not in the context of matrix operations but as an alternative to other looping constructs like while loops.

# REVIEWER 2

Building on recent work on matrix query languages, the authors propose an extension of the language MATLANG by a for-loop. They show that many natural linear algebra constructions can be expressed in this language.

The main technical result is that the new language has the same expressiveness as logspace-uniform arithmetic circuits of polynomial degree. This is a very nice and strong result,

The authors also consider various, somewhat more tractable, restrictions of the language.

As the authors point out, the for-loop implicitly introduces an order on the index space of the matrices, because iteration is over the vectors of a standard basis. For a matrix language, this is, natural, though still a bit unsatisfactory, in a relational context it may be regarded slightly problematic. I think this is outweighed by the expressivenness one gains, but it may have been the reason why people have avoided such constructions in earlier papers.

I think this is a strong paper that should be accepted.

# REVIEWER 3

## Summary

The paper introduces an extension of the query language over matrices MATLANG, which the authors call for-MATLANG, where for-loops, implemented via the use of canonical vectors, are allowed. The main premise is that plain MATLANG is not able to express important linear algebra algorithms, due to the lack of a mean to iterate over the dimensions of the input matrices. Such algorithms include computing the determinant, inverse, LU-decompositions, etc.

The authors introduce the syntax and semantics of for-MATLANG and show how the above linear algebra tasks, as well as some graph algortihms, can be implemented in the new language. The linear algebra algorithms require an external division function $f_/$ to be provided. Then, the authors embark in a more systematic expressibility analysis, by comparing the functions over matrices that are computable by for-MATLANG (without any additional external function) and uniform families of arithmetic circuits (which capture most standard linear algebra algorithms). The equivalence is shown to hold, when restricting circuits (and for-MATLANG expressions) to have polynomial degree.

Then, the authors focus on some sub-fragments of for-MATLANG, specifically the one where for-loops are only allowed to compute incremental summations of an expression, called sum-MATLANG. This sub-fragment is then extended to also allow incremental computations of the Hadamard product or the standard matrix product of an expression, leading to the languages FO-MATLANG and prod-MATLANG, respectively. They show that sum-MATLANG (when properly extended to arbitrary semirings K) is equivalent to positive relational algebra over binary K-relations. Regarding FO-MATLANG, they show that when focusing on FO-MATLANG expressions where only vectors and square matrices all of the same size are allowed, FO-MATLANG is equivalnt to weighted-logics over binary signatures. Finally, they show that prod-MATLANG strictly subsumes FO-MATLANG, and when extended with ordering information (i.e., having access to the upper triangular matrix $S_{\leq}$ with all non-zero entries equal to 1) and the external function $f_/$, the resulting language is expressive enough to compute the determinant and the inverse of a matrix. Transitive closure of an adjacency matrix is also computable, given the external function $f_{>0}$ checking whether a given real number is non-zero, is provided. No expressibility equivalences are established for prod-MATLANG.

## Review

Overall, the paper is technically solid, with non-trivial results regarding the expressivity of the new language. The modification of MATLANG via for-loops is quite natural and not very "invasive". The connection with arithmetic circuits provides a good formal argument on the capacity of the new language of expressing linear algebra constructs. I also find the sum-, FO-, and prod-fragments resonable limitations of the language. The results are correct from what I can tell.

On the other hand, I believe that on the motivations and presentation side the paper is not on par with the high standards of a PODS paper.

**(TODO)** One of the premises of the paper is to find the atomic operations

to be introduced in MATLANG, that allow to "define standard linear algebra algorithms". I totally get the point, and it is clear that continuously adding ad hoc operators to the language is not the way to go. However, if the explicitly provided premise is only to define standard algorithms, it is also fair to ask what other advantages the new language has. Implementing "simple" procedures such as 4-clique and especially the PLU-decomposition seem to require some not-so-natural expressions (and in some cases, like PLU, required to dig into some details/properties of the algorithm to implement it). One can argue that syntactic sugar can be added to aid in implementing more complex algorithms. So, I would have liked to see some discussion on this and a comparison between the new language and some of the languages mentioned in the related work section.

**Thomas:** I don't know how to properly mention that it is **a** way to go, maybo not **the** way to go, that is to be determined in the future. The main result is strong enough to consider the setting, as other reviewers mention.

**(TODO)** Space for the above discussion could be gained by pushing some more details in the appendix, such as part of the definition of$RA_K^+$ (e.g., the semantics is somehow obvious, and can be deferred to the appendix).

**(TODO)** Similarly, in the equivalence between FO-MATLANG (with all square matrices of the same size) and WL, the authors could have spent some words in describing why such a fragment on restricted square matrices (all of the same size) is interesting/useful. I also would have given some more intuition of the 4-clique expression in Example 3.3.

**Thomas:** I would maybe define the schema as a "fixed dimension schema" (accepting names suggestion). We already mention that it is for a "crisp translation between both languages". I don't know how to elaborate more, we actually state the problem and say we leave it for future work.

I also find that the paper is at times sloppy and not very formal, even at the level of the definition of the syntax and semantics of (for-)MATLANG.

**(TODO)** In particular, the semantics of (for-)MATLANG expressions is not well-defined, as no parenthesization in the syntax, or priority among the operators is specified. Hence the semantics of expressions like $A + B \cdot C$ is ambiguous.

**Thomas:** Working on this. Accepting suggestions.

**(OMITED)** Moreover, the notion of expressibility is never formally defined, especially when talking about expressing specific constructs like the 4-clique. Without clarifying this, one can (I know this is a bit extreme) e.g., express 4-clique by encoding the adjacency matrix as a natural number (in binary), and use a function f from F, that given the encoding of the matrix, checks if a 4-clique exists, placing 4-clique in $MATLANG[f]$.

**Thomas:** I think he didn't understand our approach for functions. What he says is perfectly reasonable and does not go against any of our results. For example, corollary 6.2 includes this case.

**(TODO)** Similar issues occur in other expressibility results such as Propositions 4.1, 4.2, 4.3 etc., where the instance I is never fully qualified. In this way, one can make the instance I map a dummy matrix M variable to the result of the function to be computed, and make the for-MATLANG expression be just M, making all such claims trivially true. I believe such claims should be provided

with a spirit similar to Theorem 5.3, where the variables of the expression, and the shape of the instance I are clearly defined.

**Thomas:** Working on this.

**(TODO)** Another instance of this is Corollary 5.2, where the term "equivalent" is used, without formally defining it. Considering that expressibility and equivalence results are the main task of the paper, I would make these notions and results more formal.

**(DONE)** Also, the use of symbols and operators not formally part of the language, such as the minus symbol, literal matrices (i.e., constants), etc., might deserve a footnote, mentioning that they are just syntactic sugar, and easily definable.

**(TODO)** So, I would suggest to improve presentation in the above regard, in particular w.r.t. the expressibility results.

**(TODO)** In Figure 1, the equivalences shown are a bit misleading as the paper does not show that FO-ML $\equiv$ WL in general, but only under some assumptions (square matrices schemas). The same applies to Proposition 6.7, where such crucial assumptions are specified outside (before) the claim.

**Thomas:** Maybe define something like FO-ML$_{\mathcal{S}}$ to denote FO-ML restricted to a fixed dimension schema. And something similar with weighted logics, as $WL^2$ to represent that the vocabulary has arity at most two.

**(TODO)** Finally, although I can seen in the paper where 4Clique and DP are shown to lie outside of the lower language, i.e. 4Clique in sum-ML/ML and DP in FO-ML / sum-ML, I could not find any mention (even in the appendix) that Inv and Det are actually in prod-ML + $S_{\leq}$ but not in FO-ML. This should be clarified.

**Thomas:** Working on this.

## Minor comments

- **(DONE)** pg 2, "uniform arithmentic circuits" – do you mean "uniform family"?
- **(DONE)** pg 2, "and show that two natural fragment" – fragments
- **(DONE)** pg 3, paragraph "Semantics": the schema $\mathcal{S}$ should be qualified as $(M, size)$. Same on the right column
- **(DONE)** pg 3, before Example 3.1: the notation $\mathcal{I}[v := b_i^n, X := A_{i-1}]$ is not defined for multiple variable assignments.
- **(TODO)** pg 4, when discussing how to implement arbitrary for-loop initializations, the authors conclude stating that "the result of this evaluation is equal to $[|\text{for } v, X = e_0.e|](I)$, but it appears to me that for this to be true, the instance I must also be updated, to make v have one more element, so to "sacrifice" the first one for the initialization phase, and then use the rest as expected. Moreover, due to this, the expression e must also change in the expression $(1 - min(v)) \cdot e(v, X)$, to "filter out" the first element in v, for the type to be well-defined.

  **Thomas:** Working on this.

- **(DONE)** pg 5, Section 4.1, the symbol I is used as the identity matrix, but sometimes the symbol $e_Id$ is used, please be uniform.

  **Thomas:** A small check doesn't hurt.

- **(DONE)** pg 5, right column: "provided that ithe entry" – the ith entry.

- **(DONE)** pg 5, definition of $f_/$. You might want to add a footnote to specify how to define the function when y is 0 (as any definition would be fine).

  **Thomas:** I added a footnote that says we map division by zero to zero. If you prefer to say it in a parenthesis after the definition I'm ok with it, feel free to change it

- **(TODO)** pg 7, Theorem 5.1: As discussed above, fully qualify any variables $e_\Phi$ uses, and how $\mathcal{I}$ is defined.

  **Thomas:** Working on this.

- **(TODO)** pg 8, degree of an expression: circuit families have no notion of degree, circuits do. So definint the degree of an expression as the "minimum" degree of a family does not make much sense. I would just define when an expression is of polynomial degree, as it is the only notion needed later on.

- **(DONE)** pg 8, right columns "expression are easily seen" – expressions

- **(DONE)** pg 9, title of Section 6.1: matlang – MATLANG

- **(TODO)** pg 11, "schema of square matrices" is misleading, as it makes it seem that the expressions allow arbitrary square matrices, of different dimensions, which is not the case. I would find an alternative name.

  **Thomas:** I propose the name: fixed dimension schema.

- **(DONE)** pg 12, Section 6.3: "can expressed" – can be expressed.

# REVIEWER 4

## Summary

The paper analyzes the expressive power of the language MATLANG with the additional operation to allow loops over all canonical vectors of a given dimension. The language MATLANG was presented two years ago with the goal to provide a query language for matrices. The restriction of loops to canonical vectors ensures that all loops are bounded linearly in the input size. Especially, the query language will not be Turing complete.

The main result is that MATLANG+for is equally expressive as uniform arithmetic circuits. This result is very technical and by no means trivial. It does not seem that hard to convert a given arithmetic circuit into a formula. However, one has to deal with a family of circuits that is given by a logspace turing machine. This turing machine is given a size (in unary) and then computes a circuit for the given size. Thus the actual reduction is from logspace Turing machines to MATLANG formulas. I do not see how this technicallity can be avoided.

The paper demonstrates the use of for-loops by providing MATLANG-for expressions for important linear algebra algorithms (lower-upper decomposition, inversion of matrices, computing determinants). It also discusses syntactic restrictions of the introduced loops and shows that a restricted class of for-loops is equally expressive as relational algebra over binary schemas.

This paper entirely focuses on exprissivity and leaves efficient evaluation to future work.

The prrofs are non-trivial and they look sound to me. However, it did not verify all results. Adding a reasonably powerful looping mechanism to the language is a natural extension. From a theoretical point of view with the conenctions to arithmetic circuits this seems to be the strongest of the MATLANG papers up to now and should therefore be accepted.

## Comments

**(TODO)** You have to state the operator priorities of MATLANG and for-MATLANG.

**Thomas:** Working on this. Accepting suggestions.

**(TODO)** I do not understand why an Instance has to assign values to the size symbols. All sizes are implicit by the assigned matrices. I would rather have I: M $\to$ Mat[R] just assign matrices and then require that there exist a function D: Symb $\to$ N such that each assigned matrix has the correct dimensions. This should simplify some statements.

**Thomas:** Thinking about this.

**(DONE)** I also notices that you are very sloppy when it comes to formally defining I. In most of the examples, I is not defined at all. This should be changed. Probably it would help to state that in the simple case with only one matrix symbol you write [[e]](A) to denote [[e]](I), where I is the function that assigns A to the only matrix symbol. Then you can get rid of I in the examples altogether.

**Thomas:** I only changed it in the for-MATLANG expressions, I left matrix expressions with $I$. Again, please give a small check if you can.

**(OMITED)** You point out that introducing loops also introduces order. Whether there is an order available or not is certainly an important aspect for query languages. It might be interesting in future work to look at the fragment of for-MATLANG, where the outcome of the loop may not depend on the order in which the canonical vectors are processed by the loop.

**(DONE)** When introducing the restricted variants you should point out that the variable X is not allowed inside the expression e. Otherwise sum-MATLANG will turn out to be much more powerful than you want it to be.

**(TODO)** What is your reasoning behind the name FO-MATLANG? FO is usually associated with first order logic and the name can lead to confusion. Maybe you can think of a better alternative or explain your choice of the name.

**Thomas:** Thinking about this.

**(TODO)** You should use the : latex operator instead of : in function declarations. The : is defined as an operator in latex and produces incorrect spacing.

**Thomas:** Testing and working on this.

**(DONE)** After Prop. 6.4: necessary – necessarily