PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Cristian Riveros <cristian.riveros@uc.cl>

Manuscript TODS-2022-0006

Transactions on Database Systems <onbehalfof@manuscriptcentral.com> 2 November 2022 at 19:14
Reply-To: cmj4@cs.rice.edu
To: dvrgoc@ing.puc.cl
Cc: floris.geerts@uantwerp.be, tfmunoz@uc.cl, cristian.riveros@uc.cl, dvrgoc@ing.puc.cl, cmj4@cs.rice.edu

Dear Dr. Domagoj Vrgoč,

I have received three reviews, included below along with a message from the handling Associate Editor, of your paper entitled "Expressive power of linear algebra query languages," which you submitted for publication in the ACM Transactions on Database Systems.

One thing: I'd like to apologize for how long this took. We strive to get back reviews in 4-5 months, but we failed in this case. Hopefully we make up with it here with the high quality of the reviews that we were able to obtain.

These reviews, by recognized experts in the field, have obviously been prepared with care. Based on the reviews, the recommendation by handling the associate editor, and my own assessment, I find that the paper needs to undergo a successful major revision to be acceptable for publication in TODS.

Assuming that you proceed to prepare a revision, I ask that you include comments, indicating how you have addressed each of the points raised in the reviews.

Your paper (the core of the paper, including the bibliography) cannot exceed 45 pages. Additional material can appear as an online appendix, pointed to by the paper. Both the core of the paper and the online appendix will appear in the ACM DL, but only the core will appear in print. An example from the December 2014 issue of TODS is http://dx.doi.org/10.1145/2656336, which has a 5-page online appendix.

To revise your manuscript, log into https://mc.manuscriptcentral.com/tods and enter your Author Center, where you will find your manuscript title listed under "Manuscripts with Decisions." Under "Actions," click on "Create a Revision." Your manuscript number has been appended to denote a revision.

You will be unable to make your revisions on the originally submitted version of the manuscript. Instead, revise your manuscript using a word processing program and save it on your computer. Please also highlight the changes to your manuscript within the document by using colored text.

Once the revised manuscript is prepared, you can upload it and submit it through your Author Center.

When submitting your revised manuscript, you will be able to respond to the comments made by the reviewer(s) in the space provided. You can use this space to document any changes you make to the original manuscript. In order to expedite the processing of the revised manuscript, please be as specific as possible in your response to the reviewer(s).

IMPORTANT: Your original files are available to you when you upload your revised manuscript. Please delete any redundant files before completing the submission.

Please ensure that I receive the revision within 3 months, that is, by Feb 2nd. To submit your revision, please go to https://mc.manuscriptcentral.com/tods. If I do not hear from you within this period, I will assume that you do not wish to revise the paper.

Thank you for submitting your paper to ACM Transactions on Database Systems.

Sincerely,

Chris Jermaine
Editor-in-Chief, ACM TODS

Reviewer(s)' Comments to Author:

Referee: 1

Recommendation: Accept

Comments:
(There are no comments.) [note the attachement]

Additional Questions:
Relevance to Databases: High

Significance of Contribution: High

Readability and Organization: High

Fusion of Theory and Practice: Adequate

Length (Relative to the useful contents of the paper): Just Right

Please help ACM create a more efficient time-to-publication process: Using your best judgment, what amount of copy editing do you think this paper needs?: Light

Most ACM journal papers are researcher-oriented. Is this paper of potential interest to developers and engineers?: No

Referee: 2

Recommendation: Needs Major Revision

Comments:
** Summary **

In this work, the authors propose and extension of the language over matrices MATLANG. The extension, called for-MATLANG, allows the language to perform for-loop-like iterations
over MATLANG expressions. The main motivation for this extension is that plain MATLANG is not able to express well-known linear algebra algorithms such as computing the inverse of a matrix, or computing the LU decompositions of a matrix.
The authors, after defining the syntax and semantics of the for-MATLANG, provide some preliminary insights on the expressive power of the language, showing that assuming some mild additional built-in, pointwise functions, such as division, are allowed, many important linear algebra algorithms are now expressible.

In order to formally discuss the actual expressive power of the new language, the authors compare for-MATLANG to the "language" of uniform families of arithmetic circuits of "bounded depth", which are regarded as the closest logical formalism to linear algebra. The authors show that every such a family of logarithmic depth can be simulated by a single for-MATLANG expression.
To show this, the implement a stack-based evaluation algorithm for arithmetic circuits in for-MATLANG. Then, they also show that for-MATLANG can be simulated via families of circuits (not necessarily of logarithmic depth), and they prove this result by employing somehow standard arguments.
Since the above results do not provide a complete characterization, a known result is exploited stating that families of logarithmic depth and families of polynomial degree are equivalent.
Hence, assuming for-MATLANG expressions that are equivalent to families of circuits of polynomial degree, they obtain the desired characterization:
For every function f over matrices, f is computable by a family of circuits of polynomial degree iff it is computable by a for-MATLANG expression of polynomial degree.

Unfortunately, deciding if a given for-MATLANG expression is of polynomial degree is undecidable, and the authors define different syntactic fragments of polynomial degree for-MATLANG expressions, namely sum-MATLANG (allowing only summation-based for-loops), sp-MATLANG, prod-MATLANG (allowing only product-based for-loops using the Hardmard, and standard product operator, respectively). They show sum-MATLANG and sp-MATLANG, when properly extended to matrices over a semiring K, to be equivalent to K-relational algebra over binary relations.
Regarding prod-MATLANG, they show that it is able to compute matrix inversion, given it also has access to a special matrix S_<

** Evaluation **
The paper provides a nice overview of the relative expressive power of the proposed extension of MATLANG w.r.t. existing formalisms. The most important and interesting one being w.r.t. families of arithmetic circuits of "bounded"

degree. The initial part of the paper, giving high level insights on the expressivity of for-MATLANG via implementations of key linear algebra algorithms is useful and interesting. This helps to highlight the potential of the new language.

Going to the technical results, they seem to be non-trivial, and appear to be correct, from what I can see. Having a look at the conference version, I can see that parts of the presentation have been improved, in particular the claims on expressibility are now better formally stated.

So, overall, I believe the paper contains high quality results that might be of interest to the database theory community. However, being TODS about Database *systems* and thus usually more oriented towards applied results, I wonder whether an additional (even preliminary) experimental evaluation/implementation of the language would have been needed (I will not fight to have that, though).

On the negative side, I believe that the paper is trying, in different places to oversell a bit its results. In particular, the paper makes a big deal in explaining that families of circuits are the de-facto representative logic of linear algebra, but you do not clarify if this is the case under the assumption that, e.g., the depth is bounded. Since your equivalence results are about families of bounded depth/degree, you should properly discuss to what extent, families of circuits of this kind are able to express linear algebra constructs.

Moreover, although you show via simple examples that there are for-MATLANG expressions (without bound on degree) that are not expressible via families of circuits of polynomial degree, it is not clear whether for-MATLANG (without any restriction) is able to capture all families of circuites without bound on the depth. From what I can see you do not have any proof on whether a family of circuits cannot be expressed via a for-MATLANG expression. This issue should be discussed, either via a theorem, or stating that this problem remains open.

You must be much more transparent in what you are achieving. In different places you claim that you connect for-MATLANG expressions to families of circuits, but then do it in restricted settings. I believe you should make very clear that a full characterization of the form:
for any function f over matrices, f is computable by a uniform family of circuits iff it is computable by a for-MATLANG expression
is not obtainable, or it is difficult to obtain, and left for future work. Then, you can justify restricting on circuits of bounded depth/degree.
Then, the paper should provide a discussion on what these kinds of families can actually express.

This kind of dicussion is particularly relevent in the introduction. The authors should properly clarify what is the actual capacity of for-MATLANG expressions. That is, given the characterization via polynomial degree circuits, which features do you keep, and which are you missing? For example, can you still implement Strassen's algorithm or compute discrete Fourier transformations, as argued in the introduction?
I feel that giving the circuit characterization without making explicit what these circuit families can actually do might leave the reader without meaningful "take-home messages", which I guess is what the goal of this paper is: provide key insights on what for-MATLANG can do in terms of *linear algebra* constructs (the equivalence via circuits is "just" the technical tool to convey these messages).

Similar to the above issue, I believe the title is too general. The authors study a *specific* query language, i.e., MATLANG. I understand you study fragments of it, and thus you have languageS, but I feel the title is a bit deceiving. I would make more explicit the content of the paper, specifying it is about the expressive power of MATLANG with iteration.

So, I would request the authors to expand on what for-MATLANG expressions of polynomial degree can atually achieve in terms of linear algebra constructs (e.g., exploiting the connection with families of bounded degree), or at least state what you cannot achieve. Moreover, it is important to make clear as soon as possible that a connection between general for-MATLANG and general families of circuits is not achievable (e.g., via some some formal statements, or just by making clear ths connection is left as an open problem).

Minor comments:
-when you introduce the min(v) expression for the first time (after Proposition 3.4) I would anticipate you will explain how to express it in for-MATLANG in the next section
-in page 9, definition of succ(b_i^n,b_j^n), I guess you mean [[succ(u,v)]](I), where I maps u and v to b_i^n, and b_j^n. Similarly for Prev \cdot b_i^n.
-Proposition 4.3: here you use the expression "when I assigns V to A". In similar claims, like Proposition 4.2, you do not say anything about what I does to V, and in Proposition 4.1 you use the function mat to state what is the value of V. Please make these equivalence statements more uniform.
-in different parts of the paper you say "circuits of bounded degree". In my view, this usually means that there exists a *constant* that bounds the degree of all circuits in the family, but it is not what you are considering here.
-line 22 of Algorithm 1: in the comment I guess meant that getinput(g) outputs i, and not A[i].

-Proposition 5.2 is very long, as it is defining notation in place. I would defined the required notation, such as vec(), first, and then give the claim. Moreover, you use \Sigma, which has never been defined. Do you mean {0,1}?
-Proposition 6.3: wherever you use S(e), I guess you mean type(e)
-Figure 4 is again somehow misleading, as you do not prove equivalence with those formalisms in general, but you assume e.g., bounded depth/polynomial degree, binary relations, etc. Please introduce proper notation for these restricted fragments. You could explain this notation in the caption of the figure.

Additional Questions:
Relevance to Databases: High

Significance of Contribution: High

Readability and Organization: High

Fusion of Theory and Practice: Marginal

Length (Relative to the useful contents of the paper): Just Right

Please help ACM create a more efficient time-to-publication process: Using your best judgment, what amount of copy editing do you think this paper needs?: Moderate

Most ACM journal papers are researcher-oriented. Is this paper of potential interest to developers and engineers?: Maybe

Referee: 3

Recommendation: Needs Major Revision

Comments:
The article studies the expressiveness of the MATLANG language that was introduced by the authors in earlier papers. The main contributions are depictions of how MATLANG extended with for loops can catch popular linear algebra algorithms like LU decomposition of matrices, a comparison with the expressive power of arithmetic circuits, and identification of two fragments of MATLANG that have the same expressive power as the relational algebra over semirings and weighted logic, respectively.

The technical depth of the article is huge. The constructions are quite involved, and apart from a few small mistakes correct. However, the presentation can still be improved by a lot as indicated below.
Also there is some problem in the way how the expressiveness of MATLANG is compared with arithmetic circuits.

The authors define a MATLANG expression of polynomial degree as any MATLANG expression that has an equivalent circuit family of polynomial degree. Afterwards there is the mind blowing result that this class exactly corresponds to the class of circuits of polynomial degree. Of course this result does not provide any scientific value, as it just repeats the definition. It is not clear at all how this class of MATLANG expressions looks like. Actually it is undecidable if a given MATLANG expression has a polynomial degree.

Instead there should be an (ideally syntactic) definition that is intrinsic to MATLANG. Especially it should not be necessary to refer to circuit families in order to provide a definition of polynomial degree for MATLANG expressions. If no syntactic definition is possible than a sensible semantic definition will also work.

While the small errors are easily fixable, the presentation and the comparison to circuit families of polynomial degree needs a major revision.

I did not yet check all details of the proofs. For the TM constructions I wait for a simplified description (as indicated below). I skipped some other details because of lack of time. I do not expect any big problems in these areas and therefore prefer to not delay the review further (until after my

holidays).

Comments to the authors:

Comparison to Arithmetic Circuits

I start with the comparison to arithmetic circuits. In Section 5.2 you construct MATLANG expressions that uses an input vector of the same arity as the circuit and outputs a single value, the same that the circuit will produce. Theorem 5.1 does not talk about the sizes of other matrices used.

In your construction you use square matrices and vectors of the same size as the input vector. As a result of this design decision, you must limit the construction to circuits of logarithmic depth. For the other direction however, you produce circuits of polynomial depth.

Here you introduce MATLANG expressions of polynomial degree in order to have a MATLANG class and a circuit class of equal expressiveness. Instead, I propose to change your construction of MATLANG expressions in a way that allows to handle all polynomial arithmetic circuits.

Replace Algorithm 1 with an algorithm that computes (and stores) the output values of all gates in topological order. This algorithm is way simpler and does not need a stack.

Then allow your MATLANG construction to use intermediate values of polynomial size. The main data structure is a vector that has as many entries as you have gates in the circuit. Now you can iterate over all gates (w.l.o.g. the gates are sorted in topological order) and compute all values. This is a single for loop.

Of course you still need the construction from the appendix to compute the nextgate() function, i.e. to simulate the TM that constructs the circuit. Probably it would be simpler if this TM would directly construct a vector that contains all input gates. Then you only need to call this function once for each gate and just use an additional for-loop for the aggregation.

Now you have a natural class of MATLANG expressions that exactly corresponds to arithmetic circuits. I do not see why MATLANG expressions should not be allowed to use intermediate results that have bigger arity than the input. This is a restriction that you never formulated and that is also not imposed on the circuits.

Of course, you can still discuss restricted settings, but please use sensible definitions. E.g. if you restrict the size of intermediate results in MATLANG a corresponding restriction would be on the width of the arithmetic circuit.

General notation

You should introduce and describe a consistent notation that allows to easily distinguish whether some MATLANG expressions
- construct some vector or matrix; or
- is a Boolean test (i.e., evaluates to a scalar value 0 or 1

Also, you should adopt the notation that iterator variables are easily distinguishable from other variables throughout the article. Especially if you give expressions like for col(V,y) in line 572, it would be really helpful to immediately see that y is meant to be a variable that can only take canonical vectors as values.

Actually, I would even prefer a notation where iterator variables (like i,j)

range over indices, i.e. take values from 1 to n. Then you can still write b_i if you need the canonical vector, but you could also just write V_{ij} instead of v^t * V * y, which is way easier to parse for a human. Obviously, such a notation would be just syntactic sugar.

I also suggest to consider introducing some if e_1 then e_2 else e_3 construction, where e_1 is some expression that evaluates to a scalar 0 or 1. This is used a lot in the article.

this is can be done.

You have to rename the succ and succ+ expressions, as they in fact do not test for successors. What you call successor is the less or equal relation and what you call succ+ is the less than relation. So succ could be renamed to islessorequal and succ+ could be named isless, which also directly reminds the reader that this expression is a Boolean test.

You seem to mix \to and \mapsto in function specifications in a random way. Function signatures use \to (f \colon A \to B, where A and B are domain and image of f), while \mapsto is used for concrete mappings (e.g., f \colon n \mapsto n^2). And you should use \colon instead of :, because : is treated as a division operator by Latex and thus the spacing is not correct.

LU-Decomposition

You should definitely provide some pseudocode for the LU Decomposition algorithm in order to allow a simpler comparison with your MATLANG expressions. Right now the algorithm is given as prose. Furthermore it is not even complete as the definition of c_i with i != 1 is missing.

?

Algorithm 1

Even if I suggest to scrap this algorithm altogether in favor of a more general and way simpler algorithm, I still want to list some details that are really painful to the reader:
- the aggregate function is working in a completely different way than your MATLANG construction. The MATLANG constructions is a sum over 5 expressions, which especially implies that the order of evaluation is irrelevant. However the algorithm is written in a way that the order of the statements is very important. Especially it is not the case that the five expressions correspond to five different cases of the algorithm as you claim.
- some constructions are way more complex than needed
  - simplify e_iterate by using the loop init X_1 = e_start. Then you can remove the outer "if-then_else".
  - e_pop can just pop both stacks simultaneously by just removing one row from the matrix. The complicated IdUpTo expression is not needed. Just compute

    e_pop := - G_top * G_top^T * X_k + e_Prev * V_top * e_V_top^T

    to pop both stacks. The first summand computes the delta to remove a line from the matrix and the second summand readds the pointer for the value stack. The pointer for the gate stack will be added in e_agg_(not)_last. Note that this is just a delta. Thus e_aggregate will need an additional X_k summand.
  - e_agg_prod also looks way to complicated. Why do you need IdVal? Just manipulate the single matrix entry directly:

    e_agg_prod := V^T * V_top * (V^T * (e_Prev * V_top) - 1)

    The minus 1 is to accomodate for the value that is already on the stack. No for-loop needed (hidden in the IdVal expression).
- I find it confusing that in Figure 3 you list the combined effect of two of the expressions. The figure should describe each of the expressions on

TC

TC

TC

its own. Especially as these combinations that you describe are not possible in the algorithm. The algorithm combines always three of the expressions. Also the Figure is wrong. In the upper two cases, there should be no pointer for the gate stack. And in the lower cases there should be an empty value stack according to your construction.

Also, please note that you overload n with many different meanings:
- arity of the circuit
- size of the matrices
- length of a bit vector describing a gate id

TC

Why are gate ids of linear length in the input? This would correspond to an circuit of exponential size.

TC

Proposition 5.2

First of all, add a runtime bound to the Turing machine. I do not believe that your construction can simulate every linear space machine without restrictions on the runtime.

And now: Why do you write up the proof using the most complicated type of TMs possible? For your construction it would be perfectly sufficient to have a single tape machine. Your computations take at most two IDs of size O(log(n)) as input and produce one such ID as output. This even fits on a single tape if you restrict to length n. You are assuming n > n_0 for some sufficiently large n_0 anyway.

TC

Using only a single tape makes the construction much more readable, as you can get rid of many indices.

If you change Algorithm 1 as laid out above, it could make sense to actually allow for an output "tape" that is formed like a square matrix. The head of the output tape could move in four directions (simulated by two vectors wit a single 1 entry). It is obvious (to all that know TMs) that this does not add additional power and one can easily translate the TM producing the circuit to such a TM with a square output. The advantage would be that you could directly produce the adjacency matrix of the circuit which you could use immediately in the expression that evaluates your circuit.

They didn't understand.

Also, you do not need to consider the case of small n at all. This case is already considered in the proof of Theorem 1. You can restrict Proposition 5.2 to all n greater than some n_0 (and maybe say that it also holds without this restriction). But no need to waste the space for the proof.

2.

And please try to simplify the construction of the TM. Why do you encode the position of the head in a special way if it is at the edge of the tape? Just adjust the size of the tape such that the end markers are included in the length. Then you do not need to special code this.

TC

From MATLANG to circuits:
Why do you restrict the result to MATLANG expressions, where all types only use the size symbol alpha? The construction should work in exactly the same way in the general case. OK, for uniformness you need that all sizes can be computed from the input size by a logspace TM, which results from the definition of uniform circuits where the TM just gets one input parameter. Probably you should discuss this (as it is the usual definition), but in your setting a slightly more general setting of uniform circuits would make sense.

TC

In any case, the proof needs to be reformulated in order to avoid all these pointless case distinctions on the types of the subexpressions. Do all induction cases with type_S(V)=(alpha,beta) and provide one induction step for every operation. The cases where one or both of alpha beta are 1 are special

TC

cases of the general case and subsumed by the general case. No need to do any case distinctions.

comparison with K-Relations

Your definition of the renaming operator is nonstandard. Usually this operator takes a function f that renames the variables of a given relation, i.e., the domain of f are the attributes of the relation/expression inside the operator and the image are the new (renamed) attributes. You define it the other way round. When you use the operator, you mix standard and your non-standard definition. Please stay with the established definition.

TC

The construction of algebra expression from MATLANG expressions is much more complicated than necessary. You should not do case distinctions on the types of matrices, as the general construction works independently of whether some dimension is 1 or not.

TC

Just use Rel(S)(R_V) := {row, col} for every matrix V. row and col encode the domain of the indices of the matrix, as in your construction. The only difference is, that this domain could be the singleton {1}. And you should omit the subscripts alpha and beta of row and col. They are not needed, as the domain is encoded by the relation. If you omit the subscripts you will not need to talk about types at all in most parts of the proof. The soundness of the MATLANG expression ensures that the domains of row and col are correct.

TC

To always provide a col attribute you need a new relation R_1 with attribute col and a single number 1 inside the relation. You can then change Q(v_p) to be

\sigma_{row, \gamma_p} (\rho_{\alpha -> \gamma_p}(R_\alpha) \join \rho_{\alpha -> \row}(R_\alpha) \join R_1)

TC

This construction simplifies the definition for transposition to just rename row -> col and col -> row. Also for the other operators you only need to talk about one case. And types are working flawlessly. E.g. matrix product becomes rename col -> C for the first expression and row -> C for the second expression before doing the join. However you have to explain what the join does, i.e., that it just computes the same sum as the matrix product.

TC

minor comments:
line 183: this is ugly to read. Please align at := and at if

line 216: this also should be aligned

line 234: recursion -> iteration

line 1386: Please rephrase. One could get the impression that the halting problem for linear time TMs is undecidable (which it certainly is not).

line 1435: please provide a pageref for figure 4 or mention that it is at the very end of the article.

line 1443: You have to restrict the expression e, such that it cannot use X. Otherwise Proposition 6.1 is definitely not true.

line 1449: Using your ill-defined definition of expressions of polynomial degree, the proof of Proposition 6.1 is nontrivial and cannot be omitted. You have to show that every expression of sum-MATLANG can be converted to a circuit family of polynomial degree, meeting the syntactic definition of the circuits. Showing that you cannot produce superpolynomial matrix entries is not enough.

line 1681: change Q_1 to Q_2

line 1731: again, X should not be used inside e

line 1912 (figure 4): If you adapt the constructions as indicated above, the figure is ok. Otherwise you need to specify which subclass of MATLANG is equivalent to which subclass of arithmetic circuits. E.g., right now, you only can convert MATLANG expressions that use a single size symbol alpha, as you needlessly restrict your construction.

Additional Questions:
Relevance to Databases: Very High

Significance of Contribution: High

Readability and Organization: Marginal

Fusion of Theory and Practice: Adequate

Length (Relative to the useful contents of the paper): Just Right

Please help ACM create a more efficient time-to-publication process: Using your best judgment, what amount of copy editing do you think this paper needs?: Light

Most ACM journal papers are researcher-oriented. Is this paper of potential interest to developers and engineers?: No
---

geertsTODS.pdf
52K