Revision letter
"Expressive power of linear algebra query languages"

Dear reviewers,

We wish to thank you for your thorough, insightful and constructive comments.

Please find attached our revised manuscript entitled "Expressive power of linear algebra query languages". In the process of revising the paper, we have addressed all your comments and suggestions. Please note that this resulted in several changes to the paper, which we have highlighted in red.

Cristian: For highlight an edit in the document, use the command `EDIT`.

The most important changes are the following.

1. **A clearer write-up of bla bla bla ...**

   We have completely rewritten bla bla bla ...

2. **A clearer write-up of bla bla bla ...**

   We have completely rewritten bla bla bla ...

Below we also provide a detailed list of answers and explanations to other reviewer comments, excluding those comments that we have already addressed above.

We sincerely hope that this new version of the manuscript sufficiently improves on the previous version, and that it will be considered for publication in TODS.

Best Regards,
The authors.

Reviewer 1

> (...) I would however suggest to expand a bit the paragraph on **Related work** at the bottom of page three, and to give more details; in particular this concerns the remark that classical logics with aggregation and fixed-point logic with counting (FPC) can also be used for linear algebra. This point deserves a bit more attention and more references to the literature (for instance to the work of Anuj Dawar and his co-authors). It is known that FPC can express a large collection of linear-algebraic algorithms over fields of characteristic 0 whereas there are severe obstacles in fields with finite characteristics. Incidentally, it is a pity that the authors exclusively consider linear-algebraic problems over the field of reals. I would welcome if the authors could, at least briefly, discuss the power of for-MATLANG for other fields.

ANSWER:**TODO**

Cristian: Related work about FPC: FLORIS

Cristian: About other fields: it is interesting, but the work is not pointed towards this direction. We leave this for future work. (FLORIS)

Reviewer 2

> (...) 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).

ANSWER: We fully agree with the reviewer that an experimental would be useful to have. However, we remark that this paper is a first step in understanding what can be expressed with matrix query languages, and tries to formalize the main concepts, and place them in the context of the existing literature. The paper was invited to TODS as such, and it already has more than 50 pages. We feel that experimental evaluation in fact warrants a paper of its own, and hope to follow up on this in future work. We expanded the "Conclusions and Future Work" section to better reflect this point.

> 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.

ANSWER:**TODO**

Cristian: To what I remember: We agree on lowering the tone with respect to the argument that arithmetic circuits are the de-facto for representing linear algebra. Also, we agree on adding some reference related to writing the determinant with arithmetic circuits (among other operators in linear algebra).

Cristian: CRISTIAN

> 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).
>
> (...)
>
> 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).

ANSWER:**TODO**

Cristian: With a clear statement that we left for future work the characterization of the expressive power for unrestricted for-Matlang should be enough.

Thomas: Nice. I think it is done then?

Cristian: CRISTIAN

> (...) 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.

ANSWER: We agree with this assessment and have changed the title to "Expressive power of MATLANG with bounded iteration".

> 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.

ANSWER: Good suggestion, the clarifying footnote has been moved to just before $\min(v)$ is used (page 8, line 354 of revised version).

> 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 $Prev \cdot b_i^n$.

ANSWER: Indeed, it is now corrected, but since we renamed succ to $\mathsf{isLessOrEqual}$ it now says $[[\mathsf{isLessOrEqual}(u,v)]](\mathcal{I}[u := b_i^n, v := b_j^n])$ (page 9, line 427 of revised version).

> 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.

ANSWER: Thanks for noticing. All the statements use now “when I assigns V to ...”. This was fixed in Proposition 4.1 (page 13, line 621 of revised version) and in Proposition 4.2 (page 14, line 667 of revised version).

> 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.

ANSWER: Indeed. At some point we started using “circuits of bounded degree” to mean that is “tractable” (in particular, polynomial). As you say, it is not the usual meaning so we change it to polynomial degree to be precise. This has been changed, it happened only once (page 19, line 901 of revised version).

Cristian: TO DO: extend this answer explaining what happen.

> Line 22 of Algorithm 1: in the comment I guess meant that $getinput(g)$ outputs i, and not $A[i]$.

ANSWER: Indeed. It is now fixed (page 21, line 1004 of revised version).

> 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 Σ, which has never been defined. Do you mean $\{0, 1\}$?

ANSWER: Good catch, $vec()$ and Σ have been properly defined before the statement (page 23, line 1080 of revised version).

> Proposition 6.3: wherever you use $S(e)$, I guess you mean $type(e)$.

ANSWER: Yes, it is meant as a shorthand, but it is not precise so it is now fixed everywhere: $\mathcal{S}(e)$ was replaced by $\mathsf{type}_{\mathcal{S}}(e)$.

> 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.

ANSWER:**TODO**

Floris: We could write: $\texttt{sum-ML}|_{sq} \equiv \texttt{RA}^{+}_{K}|_{sq,bin}$, $\texttt{sp-ML}|_{sq} \equiv \texttt{WL}|_{sq,bin}$ and $\texttt{for-ML}|_{sq,poly} \equiv \text{Arithmetic Circuits}|_{sq,poly}$ and describe in the caption that *sq*: square matrices, *bin*: binary schemas, *poly*: poly degree. Not implemented yet!

Thomas: I agree.

Cristian: CRISTIAN will change the figure adding daggers.

Reviewer 3

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

ANSWER:**TODO**

> 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.

ANSWER:**TODO**

Thomas: We need to discuss about for-MATLANG expressions of polynomial degree.

Cristian: DOMAGOJ: remove theorem 5.6 and the definition of for-Matlang of polynomial degree.

Domagoj: As discussed with Cristian, I tried to rephrase the whole section and change the definition to an operational one given by our results. It is not very satisfying, and perhaps some more discussion is needed. But it should be very clear what is done at this point.

Thomas: I think it is ok now.

Cristian: IMPORTANT: we can give a semantic definition, not related to the circuit, by saying that the output of the expression is bounded by a polynomial. I will implement this.

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.

ANSWER:**TODO**

Thomas: The size of the main matrix is $n \times n$. I added it but it looks somewhat off so maybe it adds just noise.

Thomas: "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."
This maybe is worth a further discussion because the reviewer is right, but it is not a problem because we say afterwards that any function computed by a uniform circuit family of polynomial degree (and polynomial depth) can also be computed by uniform circuit family of logarithmic depth,

Thomas: I believe I do not understand the reviewer's proposal for the refactor of algorithm 1.

Cristian: CRISTIAN: Write a technical and diplomatic answer.

> You should introduce and describe a consistent notation that allows to easily distinguish whether some MATLANG expressions (1) construct some vector or matrix; or (2) is a Boolean test (i.e., evaluates to a scalar value 0 or 1).

ANSWER: Thanks for the suggestion. The expressions that result in a boolean value are now explicitly named (isLess, isLessOrEqual,etc.), whereas the expressions that build matrices or vectors are kept as e_{name}.

> 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.

ANSWER: Yes, thanks for noticing. We now mention that uppercase letters always refer to matrices (of any size) and lowercase letters always refer to iterator variables (page 6, line 261 of revised version).

Cristian: FLORIS will improve it.

> 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.

ANSWER:**TODO**(maybe not).

Thomas: The former is a good idea, but besides replacing using V_{uv} instead of $u^t * V * v$, there are no major advantages, while it has major impact in rewriting expressions. Will do if time allows it.

Cristian: We CANNOT make this change. It could rather confuses the notation when the vector is used as an index, or as a vector (for example when you sum all cannonical vectors).

Cristian: FLORIS will give a diplomatic answer.

> 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.

ANSWER: Yes, it is indeed handy and it is now defined (page 11, line 492 of revised version). Thanks for the suggestion.

Cristian: DOMAGOJ: Will improve the introduction of this if-then-else. Add brackets.

Domagoj: 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.

ANSWER: Indeed is clearer with the new names: isLessOrEqual and isLess (page 9, line 422 and 432 of revised version).

Cristian: Use Camel case like isLessOrEqual.

Thomas: Done.

> 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., $f\colon n \mapsto n^2$).

ANSWER: Thanks for noticing the mixups. They are now corrected, the function signatures (page 32, line 1554) and the concrete mappings in the renaming operator: $\rho_{a\mapsto b}$ (pages 33 and 34).

> And you should use `\colon` instead of :, because : is treated as a division operator by Latex and thus the spacing is not correct.

ANSWER: Thanks for the suggestion. The change was made and the formulas look better. Most problems are with :=, so it was replaced by `\coloneqq` command.

> 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 $i \neq 1$ is missing.

ANSWER: Good suggestion, the prose can be confusing on its own. A pseudocode has been added (Algorithm 1).

Cristian: FLORIS will improve the presentation of the algorithm.

> Algorithm 1: 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.

ANSWER: Correct, it needed further clarification: the expressions e_{extend} or $e_{\mathsf{aggregate}}$ use X_k and *return* an updated X_k for the next iteration, by building parts of the to be returned matrix independently. In the case of e_{extend} it is straightforward because each case is totally independent of each other. The case of $e_{\mathsf{aggregate}}$ is a bit unclear because there are multiple subcases over the sum of the 5 subexpressions. With the new comments and updated figure, it should be clear that each subexpression affects specific and independent parts of the matrix that will be returned and thus $e_{\mathsf{aggregate}}$ is indeed a commutative expression in the sense that it does not matter the order in which the subexpressions are computed and summed because they have no common nonzero entries.

Cristian: CRISTIAN: Check this and improve the answer.

> Algorithm 1: simplify $e_{iterate}$ by using the loop init $X_1 = e_{start}$. Then you can remove the outer "if-then-else".

ANSWER: Nice catch. It is now corrected as suggested (now in Algorithm 2).

> Algorithm 1: 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 reads 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.

ANSWER: It is a good idea if the algorithm is based in deltas, but it is not the case. As It is now depicted clearly, the idea is for each expression to compute independent parts of the matrix that will be outputed as X_k for the next iteration so they can be summed in any order. The change to a delta approach means a refactor over all expression to take into account possible interactions between common entries.

Cristian: CRISTIAN: similar that previous comments about deltas.

> Algorithm 1: 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 -1 is to accomodate for the value that is already on the stack. No for-loop needed (hidden in the $IdVal$ expression).

ANSWER: Again, it is a good idea if the overall approach was intended as a delta approach.

Cristian: CRISTIAN

> Algorithm 1: 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 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.

ANSWER: Correct. It is now fixed and done as suggested.

Cristian: FLORIS: to improve the figures.

> Algorithm 1: Also, please note that you overload n with many different meanings: (1) arity of the circuit, (2) size of the matrices, and (3) length of a bit vector describing a gate id.

ANSWER: It is confusing but somewhat necessary to encode all the information of the circuit Φ_n into one matrix of dimensions $n \times n$ (note that the (1) arity of the circuit is then equal to the (2) size of the matrices involved). To make a more precise explanation of the (3) length of a bit vector describing a gate id, now given a circuit Φ_n of logarithmic depth (and polynomial size), every gate of Φ_n is associated with an id such that $\mathtt{id} \in \{0,1\}^d$, with $d = k\lceil \log(n) \rceil$ for some k. Then we assume that this $\mathtt{id}$ is of size n without loss of generality since circuit Φ_n has polynomial size and thus there exists k such that $\log(\mathsf{gates}(\Phi_n)) \leq k\lceil \log(n) \rceil \leq n$.

Cristian: THOMAS: will change this? Floris said: "Be careful"

Thomas: I think it is ok now.

> Algorithm 1: Why are gate ids of linear length in the input? This would correspond to an circuit of exponential size.

ANSWER: Indeed, now each $\mathtt{id}$ of a gate is of size $d = k\lceil \log(n) \rceil$ and then it is assumed to be of size n without loss of generality since if circuit Φ_n has polynomial size then there exists k such that $\log(\mathsf{gates}(\Phi_n)) \leq k\lceil \log(n) \rceil \leq n$.

Cristian: CRISTIAN: clarify this.

Thomas: I think it is ok now.

> 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.

ANSWER: Good observation. The runtime bound if the Turing machine is part of the result but was left out to keep the statement simple. Now we added the full statement.

Cristian: THOMAS will move the statement in the appedix to the body.

Thomas: Full statement added

> Proposition 5.2: 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. Using only a single tape makes the construction much more readable, as you can get rid of many indices.

ANSWER:**TODO**

Cristian: CRISTIAN

> Proposition 5.2: 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.

ANSWER:**TODO**

Cristian: CRISTIAN

> Proposition 5.2: 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.

ANSWER:**TODO**

Cristian: FLORIS

> Proposition 5.2: 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.

ANSWER:**TODO**

Cristian: CRISTIAN

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.

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 $\texttt{type}_S(V) = (\alpha, \beta)$ and provide one induction step for every operation. The cases where one or both of alpha beta are 1 are special cases of the general case and subsumed by the general case. No need to do any case distinctions.

ANSWER:**TODO**

Thomas: This solves if we use the new names for the squared fragments.

Cristian: DOMAGOJ: We say at the beginning to keep this simple we use square matrix.

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.

ANSWER: Indeed, now we use ρ_f where $f\colon X \to Y$. And for $a \in X, b \in Y$, we use $\rho_{a \mapsto b}$ to say that attribute with name a will have now name b.

Cristian: CRISTIAN: Check it.

> Comparison with K-Relations: 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.

ANSWER: Indeed, thanks for noticing. The proof of Proposition 6.3 is now simplified as suggested (page 33, line 1576 of revised version).

> Comparison with K-Relations: 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.
>
> 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
>
> $$\sigma_{row,\gamma_p}(\rho_{\alpha->\gamma_p}(R_\alpha)\texttt{join}\rho_{\alpha->row}(R_\alpha)\texttt{join}R_1)$$
>
> This construction simplifies the definition for transposition to just rename row $\rightarrow$ col and col $\rightarrow$ 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 $\rightarrow$ C for the first expression and row $\rightarrow$ 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.

ANSWER: Good suggestions. The changes were made accordingly in the proof of Proposition 6.3 (page 33, line 1576 of revised version).

Summary: use $Rel(S)(R_V) := \{row, col\}$ for every matrix V; use R_1 in the contruction of the relation corresponding to v; do not do case distinctions based on sizes; explain what the join does.

Cristian: FLORIS

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

ANSWER: Indeed. It is fixed (page 5, line 199 of revised version).

> line 216: this also should be aligned.

ANSWER: Also true. It is fixed (page 5, line 229 of revised version).

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

ANSWER:**TODO**

Thomas: We need to discuss about for-MATLANG expressions of polynomial degree.

Cristian: CRISTIAN: It is confusing. Clarify this in the paper.

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

ANSWER: Good suggestion. The pageref was added (page 31, line 1476 of revised version).

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

ANSWER: Yes, it is now clarified (page 31, line 1485 of revised version).

> 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.

ANSWER:**TODO**

Thomas: We need to discuss about for-MATLANG expressions of polynomial degree.

Cristian: DOMAGOJ: this is related with the change about for-matlang of polynomial degree.

Domagoj: We need to check if the results follows easily now from the definition.

> line 1681: change Q_1 to Q_2.

ANSWER: Yes, it is now corrected (page 36, line 1721 of revised version)

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

ANSWER: Yes, it is now clarified (page 37, line 1775 of revised version).

> 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.

ANSWER:**TODO**

Thomas: This solves if we use the new names for the squared fragments.

Cristian: CRISTIAN: the expert in figures.

References