We thank the reviewers for the time and effort invested in the detailed reviews and the valuable suggestions and comments to improve the quality of the paper. Below we reply to the individual points of the reviewers.

1 Rebuttal to Review 1

> *There should, however, be some discussion somewhere in the paper on the relationship between path length and KB size. Even if a KB uses a non-parallelizable ontology language this doesn't matter much if the paths that cause problems are short compared to the size of the KB. The non-parallel aspects of path processing of short paths will not be noticable with a small number of threads (certainly up to hundreds) and maybe not even for very many threads (because of general overhead).*

We added the parallel complexity analysis for Algorithm $\mathsf{A}_{\rm bsc}$ and Algorithm $\mathsf{A}_{\rm opt}$ in Section 3.3 and Section 3.4 respectively. Based on the analysis and Theorem 1, we have that the parallel complexity of Algorithm $\mathsf{A}_{\rm bsc}$ depends only on the graph depths, while the parallel complexity of Algorithm $\mathsf{A}_{\rm opt}$ depends on both of the graph depths and the sizes of inputs.

We added the discussion of the graph depths for real ontologies in Section 6.3. From the evaluation, we observed that the graph depths were far less than the sizes of the input ontologies. Thus, the reasoning times of the test ontologies were mainly determined by the input sizes.

> *The second sentence of Section 3 is badly stated and misleading. It is definitely true that for a Datalog program $\langle R, I\rangle$ the rule set R is fixed. However, the initial facts I are also fixed.*

We have rephrased the beginning of Section 3 and made it clear that the rule set is fixed for a class of Datalog programs. We also revised the similar presentations in Section 2.

> *It is not clear whether the classes of Datalog programs considered in the paper have to share the same rule set. The discussion previous to Definition 1 seems to so indicate, but there is nothing in the definition that says so.*

We modified Definition 1 to make it clear that the Datalog programs in a class share the same rule set.

> *There is no definition of what the parents of a node in a graph are. However, it is quite obvious what the definition should be.*

We added the definition of the parents of a node in a materialization graph in Definition 2.

Example 6 does need a bit of revision. It is not the case that Oex3 cannot be materialzed tractably in parallel. Instead there has to be a class of ontologies that cannot be so materialized.

We did the revision for Example 6 (see the paragraph following Example 6), and Example 1 (see the last paragraph of Section 3.2).

It would be much better if the ontology names corresponded to the example numbers, i.e., the ontology in Example 6 was Oex6.

We made the ontology subscripts correspond to the example numbers for Example 5, Example 6, Example 7 and Example 9.

The notion of a simple concept is very much tied to the algorithm developed in the paper. How often do actual ontologies have constructs that could make them intractable that are rended tractable by having simple concepts? If there are none, then this class is uninteresting, particularly as the paper makes such strong claims about worrying about particular ontologies. This is different from an ontology only containing simple concepts.

We investigated many real ontologies from different data source. These ontologies cover several domains and only contain simple concepts (see Section 6.1).

In this work, we focused on exploring the properties that make ontologies tractable in parallel. Thus, we did not study the problem of making ontologies tractable in parallel by having simple concepts and its influence. We will study this problem in the future work.

The paper is all about parallel tractability of materialization, both theoretical and practical. However the paper gives insufficient attention to RDFS, which is the main ontology language used in practice, and particularly for large knowledge bases. Even though RDFS is laughably simple it does not have parallel tractability of even entailment. The paper should better examine parallel tractability in the context of RDFS or RDFS with minor extensions.

Of course, RDFS does not fit into the approach of the paper, as it does not a-priori separate classes and individuals. The paper needs to discuss this issue, as it prominently mentions RDFS.

We used a new section (Section 4.4) in the revision to discuss why RDFS does not have parallel tractability of entailment.

The paper does a bad job of describing why materialization in RDFS is not parallizable. RDFS KBs have an infinite number of consequences, so it is certainly not possible to materialize all of them quickly. There are, however, well known tricks that can be used to *generate a finite representation of this infinite set of consequences. However, even this cannot be done quickly in parallel. In fact, even RDFS entailment cannot be done quickly in parallel, because RDF-S allows for a kind of twisted interaction (which I think it similar to twisted paths in the paper) when computing subproperty relationships. Of course, having subproperties of rdfs:subPropertyOf is not a usual thing to do in RDFS. The paper needs to be much more clear on its relationship to RDFS.*

We gave an example that RDFS entailment may lead to the situation of path twisting in Section 4.4 of the revision. We also discussed in what cases RDFS has parallel tractability of entailment.

The paper mentions YAGO early on but does not indicate why YAGO admits good performance for parallel reasoning. YAGO uses a very simple ontology language - RDFS plus two simple extensions. It is thus a prime candidate for close examination to show how practical results differ from worst-case theoretical ones. However, the paper does not have any examination of reasoning on YAGO. Such an examination is very much called for. Similarly an examination of the performance of other large RDFS-based KBs, such as DBpedia, would be very useful. If YAGO is not benchmarked then it needs to be dropped entirely. Similarly for LUBM.

We extended the discussion for YAGO and LUBM in Section 6.1. We added the evaluation for these two kinds of ontologies in Section 6.2.

The benchmarking part of the paper is very limited and extremely hard to follow. Only eight ontologies were benchmarked. Why were these eight chosen? Only considering at most eight threads is much too constricting.

The eight ontologies are selected since they satisfy the properties for parallel tractability. We evaluated them to show the difference between the ontologies of tractable and intractable in parallel.

More detail is needed on how the KBs were constructed. Chain length is a very important aspect of the KBs, but nothing is said of how the instances in the KB are connected together. It is not stated which ontologies belong to which DL.

In order to make the experiments close to real scenes, we did not construct these eight ontologies for some purpose, e.g., to make the graph depth deeper. Thus, these ontologies acted like common ones.

Figure 5 is very misleading because it uses different scales for the two systems. There are several immediate questions that arise from this

Figure that are poorly handled in the paper. First, why does going from one thread to two make such a difference for ParallelDHL? Without a convincing explanation for this surprising aspect of the benchmarking it is hard to trust any of the rest of the benchmarking. The other surprising aspect of Figure 5 is that RDFox speeds up so little. There is a short explanation of why this is so, but I would have liked to see a longer one.

We checked that ParallelDHL costed a large amount of time for computing the relation S_{rch} when only one thread being allocated. When more than one thread is allocated, ParallelDHL has a better efficiency. This also resulted in the difference of the speedup between RDFox and ParallelDHL.

The computation of the speedup numbers is wholely unexplained. Why should this number have any importance?

We used the speedup numbers to show the distinguishes of reasoning performance between the ontologies in $\mathcal{D}_{dhl(\circ)}$ and $\mathcal{D}^{-}_{dhl(\circ)}$.

There is a claim that the benchmarking distinguishes between the Ddhlo and D-dhlo. However, Figure 5 doesn't really show this well, if at all. For example, why should processing slow down with more threads at all? Further, it appears that even for several of the more well-behaved ontologies that there is very little speedup between 6 and 8 processors.

We did not construct the eight ontologies for some purpose. Thus, these ontologies acted like common ones. From Figure 5, the distinguishes between the ontologies in $\mathcal{D}_{dhl(\circ)}$ and $\mathcal{D}^{-}_{dhl(\circ)}$ were not shown for all the test ontologies.

The benchmarking should be redone with more ontologies and with more threads. As well, there needs to be better analysis of where parallelism is failing. For example, if chains are short, then there should be no problem in achieving near-perfect thread utilization, ignoring memory contention problems.

We added the detailed analysis for LUBM and YAGO and the influence of graph depths in the revision.

Wording and structural problems:

The paper has so many grammatical errors near its beginning that I gave up marking them after the first two pages.

The paper has been carefully proof-read and hopefully all wording and structural issues are fixed now.

"parallely tractable" is very grating. It would be better to rewrite sentences to not use it.

TODO: once done, check whether indeed all occurrences are removed
We have removed all occurrences of "parallely tractable" and rephrased the sentenced to use "tractable in parallel" or "parallel tractability" instead.

> *The abstract of the paper is much too long. An abstract is supposed to be one not-long paragraph. Instead it is here two long paragraphs. The abstract needs to be cut down to a suitable size before any publication.*

Indeed, the abstract has been shortened significantly now.

2 Rebuttal to Review 2

> *First, in the discussion under Lemma 1, it says $|P^*|$ is polynomial in the size of P, which is imprecise, as $|P^*|$ is polynomial in the size of I but not necessarily in the size of R. This is not a big issue if one considers NC for data complexity. What is less straightforward and more of concern is the statement that Step 2 costs only one time unit. As Step 2 involves checking the applicability of $B_1, \ldots, B_n \to H$ in G and updating G, it depends on the size of G and the size of G is not a constant w.r.t. that of I. Footnote 4 suggests checking applicability of the rule can be done in constant time via an index. What about updating G and maintaining the index?*

We rewrote the paragraph under Lemma 1 to analyze the parallel complexity of Algorithm A_{bsc}. We made it clear that $|P^*|$ is polynomial in the size of I for a class of datalog programs.

The statement that Step 2 costs only one time unit is based on the context of one processor. Since one processor is allocated one rule instance (see the first paragraph of Algorithm A_{bsc}), the procedure can be completed in constant time units.

The constant time of checking applicability by a processor is based on an assumption: for any datalog program $P = \langle R, \mathbf{I} \rangle$, any substitution of some atom and any rule instance in P^* can be mapped to a unique memory location; further, a one-to-one relation can be established between processors and rule instances. Under this assumption, a processor can check the applicability of its corresponding rule instance and access the state of an atom occurring in this rule instance in constant time. This assumption also applies to analyzing the computation of transitive closures [1] and the problem of boolean matrix multiplication [2]. Without this assumption, the program costs at least linear time to load the inputs. Thus, the requirements of the NC algorithms cannot be satisfied.

We gave detailed information about the above assumption before Algorithm A_{bsc}.

The computation of the rch relation S_{rch} (†) looks problematic, as it requires "H has been added to G". If it were the case, then in Example 4 the S_{rch} would always be empty, as none of the rule heads H would be added to G. Again, the complexity analysis of A_{opt} is unclear and possibly flawed. Step 2 uses an NC algorithm to compute the transitive closure of S_{rch}, which runs in poly-logarithmic time w.r.t. the size of S_{rch} not necessarily so w.r.t. the size of I. What is the size of S_{rch} w.r.t. that of I? Similarly as above, it involves updating G, which depends on the sizes of both G and S_{rch}. Would it be still in poly-logarithmic time?

The requirement "H has been added to G" in the description of computing the rch relation S_{rch} (†) is indeed incorrect. It should be "H has not been added to G". We fixed this error.

We rewrote the paragraph under Lemma 2 to give a more detailed analysis of the parallel complexity of Algorithm A_{opt}. In Step 2 of Algorithm A_{opt}, an NC algorithm is used to compute the transitive closure of S_{rch}. It can be checked that the scale of S_{rch} is polynomial in the size of inputs. The total computing time is still in poly-logarithmic time (see the detailed analysis in the revision).

Finally, the algorithms assume each processor handling a rule instantiation. I wonder how it is implemented in ParallelDHL.

We added the discussion of how each processor handles rule instantiations at the beginning of Section 6.2.

I list some relatively minor issues regarding presentation and technical details:

- p20, it mentions a group of materialisation rules in YAGO which belong to D_{dhl}. It deserves a clarification on what these rules are like and why they do not fall into existing parallel tractable classes.
- p20, Table 2, it should mention how these ontologies were selected.
- p21, the discussion on line graph lg1 is confusing (i.e. "line-2 stays higher than line-6, but lower than line-4"). Maybe I was looking at the wrong place.
- p25, the section heading "Reference" occur twice.

TODO: remove the quote once the issues are fixed.
All minor issues have been fixed.

References

[1] E. Allender, Reachability problems: An update, in: Proc. of CiE, 2007, pp. 25–27.

[2] R. Greenlaw, H. J. Hoover, W. L. Ruzzo, Limits to Parallel Computation: P-Completeness Theory, Oxford University Press, New York, 1995.