Statement Addressing the Comments and Remarks of the Reviewers

We thank the anonymous reviewers for the time and attention that you have spent our paper, and for the highly useful comments and remarks. We explain below how we addressed each of your remarks.

Reviewer 2

> The authors present a method to compress first-order resolution proofs, extending previous work for propositional resolution. Regrettably, their main result is incorrect (see the attached PDF). I have the impression that it is possible to repair the bug; therefore my recommendation is "revise", rather than "reject".

Thank you for the counter-example.

Our first step to repair this bug was to run our implemented algorithm on the given example. Our algorithm correctly identified that the proof should not have been regularized. Therefore, we identified that the bug was in the description of the definitions in the paper, and not in the implementation.

In the process of fixing the discrepancy between the implementation and the theoretical definitions, we realized that we could simplify the definitions slightly, and this led us to modify the implementation as well.

We have re-run all the experiments and produced new charts. The experimental conclusions remained the same.

> I would appreciate, though, if the authors could provide a formal theorem stating the main result and a formal proof for it in the revised version, rather than relying on the reviewers' and their own intuition.

We have added a theorem and a proof, as requested.

Our original submission did not contain a proof of this result because of the following reasons:

1) We did not consider the theoretical correctness of a particular lifting to be our main result. Our main result is experimental and exploratory. We have pushed the lifting as far as we could. We could have defined more partial liftings that would have been easier to

prove theoretically. But we preferred instead to push towards more complex algorithms that can potentially detect and remove more irregularities. In this regard, we reproduce below some paragraphs of our reply to Reviewer 1 that might interest you as well:

Satisfying a request of the other reviewer, we now have a proof that removing "strongly regularizable" nodes is successful (in the well-defined sense of theorem 4.4).

*We should keep in mind that we are talking about *partial* regularization, and hence success is relative (even in the propositional case [*]). It depends on how partial we want to be. The less we want to regularize (i.e. the stronger the condition that we impose to consider a node potentially "regularizable"), the easier it is to prove that it is ok to regularize a "regularizable" node. In practice, finding the weakest possible condition is not necessarily desirable, because this condition may be more expensive to check. Furthermore, in practice, even if we had a condition that is too weak (i.e. for which an analogous of theorem 4.4 could not be proven), that condition might still be useful, if it served as an efficient way of detecting regularizable nodes efficiently and with a sufficiently high precision. In cases when the condition failed (giving a false positive when detecting a regularizable node and forcing the algorithm to fail to construct a shorter proof), we could simply return the original proof. There would be no harm.*

[*] Compare, for instance, RecyclePivots and RecyclePivotsWithIntersection (in the propositional case). They differ from each other on the condition used to detect regularizable nodes. Something similar happens for LowerUnits and LowerUnivalents. They differ on the strength of the condition used to detect nodes that can be lowered.

2) In a special issue for "Practical Aspects of Automated Reasoning (PAAR 2016)" and "Artificial Intelligence and Theorem Proving (AITP 2016)", which are two events where experimental papers are welcome, we thought an experimental paper would be welcome.

Reviewer 1

> Therefore, this investigation is only a first stage, and one may expect Follow-up papers adding further inference rules, up to all those of a standard inference system for first-order logic with equality. The authors should discuss the relevance of proofs made only of binary resolution and factoring steps: are they relevant only as a building block or in themselves?

We see it as the main building block. As we explain in the last paragraph of the conclusion, state-of-the-art theorem provers use restrictions and extensions of resolution, and we argue that dealing with these restrictions and extensions of resolution should be conceptually simple, at least in comparison with the main challenges addressed in this paper.

In order to prevent false expectations until the conclusion, we have added a paragraph at the end of the introduction, warning the reader that state-of-the-art provers do not use the pure first-order resolution calculus on which our paper focuses and that we discuss this briefly in the conclusion.

The added paragraph is reproduced below:

"It is important to emphasize that this paper targets proofs in a pure first-order resolution calculus (with resolution and factoring rules only), without refinements or extensions, and without equality rules. As most state-of-the-art resolution-based provers use variations and extensions of this pure calculus and there exists no common proof format, the presented algorithm cannot be directly applied to the proofs generated by most provers, and even SPASS had to be specially configured to disable SPASS's extensions in order to generate pure resolution proofs for our experiments. By targeting the pure first-order resolution calculus, we address the common theoretical basis for the calculi of various provers. In the Conclusion (Section 7), we briefly discuss what could be done to tackle common variations and extensions, such as splitting and equality reasoning. Nevertheless, they remain topics for future research beyond the scope of this paper."

> The paper does not motivate proof compression: why is it good to compress proofs?

The audience we expect is people who are already interested in proof compression (and preferably who already know the propositional version of the algorithm we are lifting), especially because this is not an easy paper and it is not intended to be a good entry point for those who need to be motivated to the topic of proof compression. Therefore, we thought that it would be redundant to have a lengthy discussion about motivations and to repeat details that can be found in other, easier, papers.

Nevertheless, we added the following sentences providing more motivation in the introduction:

"[...] and the most efficient provers do not necessarily generate the shortest proofs. One reason for this is that efficient resolution provers use refinements that restrict the application of inference rules. Although fewer clauses are generated and the search space is reduced, refinements may exclude short proofs whose inferences do not satisfy the restriction.

Longer and larger proofs take longer to check, may consume more memory during proof-checking and occupy more storage space, and may have a larger unsat core, if more input clauses are used in the proof, and a larger Herbrand sequent, if more variables are instantiated. For these technical reasons, it is worth pursuing efficient algorithms that compress proofs after they have been found. Furthermore, the problem of proof compression is closely related to Hilbert's 24th Problem, which asks for criteria to judge the simplicity of proofs. Proof length is arguably one possible criterion for some applications."

> In the first paragraph of the introduction the authors write that "the best, most efficient provers do not necessarily generate the best, least redundant proofs." However, there is no discussion in the paper on what it means for a proof to be "good" and therefore "best", or "Redundant" and therefore "less" or "least redundant." The paper takes pretty much for granted that eliminating nodes is good. Why? One gleans from this that the underlying measure is proof length, measured by number of nodes, but why do we adopt proof length as measure?

We addressed this concern by replacing "best, least redundant" by "shortest" in the paragraph that we added in the introduction. See above.

> What is the balance of advantages and disadvantages of proof compression? Could it be that we gain a shorter proof at the expense of something else? Readability? Other features? If not, why? Probably this balance depends on the application (e.g., Proof-carrying code? Other kinds of certificates?), but the authors neither mention nor discuss them. Proof presentation is an important issue in theorem proving: what is the impact of proof compression on proof presentation?

We don't see any disadvantage that could occur by using FORPI (the algorithm described in this paper). The number of clauses reduces. The number of literals per clause reduces (except in one special case, when the deleted parent of an irregular node is a unit clause). Fewer instantiations are made, and therefore terms become shorter. Consequently, proof checking time reduces. If we are lucky, also the unsatisfiable core reduces. We now discuss this in the paragraph that we added.

Some other proof compression algorithms may have disadvantages. For instance, the LowerUnits algorithm reduces proof length, but increases clause length.

> The paper does not really explain the meaning of regularization, not even in the propositional case. Sure, regularization removes Nodes and therefore reduces proof length measured as number of nodes, but is there anything else? An intuition of this reader is that regularization may be connected to the level saturation strategy for propositional resolution:
> impose an ordering A_1 < A_2 < A_3 < ... on the propositional atoms, and then first do all resolutions on A_1, then all resolutions on A_2, and so on.

Your intuition (that there is a connection) is right. This connection can be clarified as follows:

The level saturation strategy will only generate fully regular proofs (i.e. proofs with no irregularity). And, therefore, applying our regularization algorithm to a proof obtained using the level saturation strategy would provide no compression. However, it would be wrong to conclude from this that the effect of our regularization algorithm would be the same as the effect of using the level saturation strategy.

It is known that fully regular resolution proofs can be, in the worst case, exponentially longer than unrestricted resolution proofs, and the level saturation strategy is susceptible to this exponential blow-up, because it always results in a *fully* regular proof. Our algorithm, on the other hand, performs *partial* regularization, removing irregularities only when the removal does not require duplication of nodes. Consequently, our regularization algorithm never increases proof length and is immune to the exponential blow-up.

> since the level saturation strategy shows that we can get rid of propositional atoms one at a time, a proof that resolves on an atom and then returns to resolve on the same atom further down on the same path may be doing something wasteful that regularization removes. Is this the idea?

Roughly, yes. But note the comment above: fully regular resolution proofs can be, in the worst case, exponentially longer than unrestricted resolution proofs. Therefore, irregularities are not always wasteful.

Because of this exponential blow-up, we would advise against any proof search strategy that always results in fully regular proofs.

> What is then the meaning of the proposed notions for regularization in the first-order case?

It is still essentially the same. A node N is irregular if there is another node N' on the path from N to the root such that N and N' have the "same" resolved atom. The tricky challenge is to define what the "same" means in first-order logic. One trivial solution would be to require them to be syntactically equal, but this would still be an essentially propositional notion of irregularity. We explored a few alternatives, relaxing this requirement and requiring only weaker conditions (related to the mutual unifiability of the resolved atoms).

> If the intuition that there is a connection between level saturation and regularization is correct, it is not surprising that lifting regularization to first order turns out to be complicated.

Yes, it does become inherently complicated, and we believe this justifies the heavier notation that we have chosen.

> Section 4 presents a series of attempts, and it is not clear whether one and which one is successful.

Satisfying a request of the other reviewer, we now have a proof that removing "strongly regularizable" nodes is successful (in the well-defined sense of theorem 4.4).

We should keep in mind that we are talking about *partial* regularization, and hence success is relative (even in the propositional case [*]). It depends on how partial we want to be. The less we want to regularize (i.e. the stronger the condition that we impose to consider a node potentially "regularizable"), the easier it is to prove that it is ok to regularize a "regularizable" node. In practice, finding the weakest possible condition is not necessarily desirable, because this condition may be more expensive to check. Furthermore, in practice, even if we had a condition that is too weak (i.e. for which an analogous of theorem 4.4 could not be proven), that condition might still be useful, if it served as an efficient way of detecting regularizable nodes efficiently and with a sufficiently high precision. In cases when the condition failed (, giving a false positive when detecting a regularizable node and forcing the algorithm to fail to construct a shorter proof), we could simply return the original proof. There would be no harm.

[*] Compare, for instance, RecyclePivots and RecyclePivotsWithIntersection (in the propositional case). They differ from each other on the condition used to detect regularizable nodes. Something similar happens for LowerUnits and LowerUnivalents. They differ on the strength of the condition used to detect nodes that can be lowered.

> It is too short and too technical. Do the algorithms' names help in an abstract? Envision a reader who considers reading this article without having read previous papers on regularization.

We have extended the abstract to provide more context, a brief description of partial regularization, and full citations to ensure the abstract is self-contained. The algorithm names help to make the contributions clear (e.g which previous algorithm was generalized, and which algorithm is also used in the experiments).

> The first sentence on first-order theorem provers gives references for Beagle, E, SPASS, SPASS+T, and Vampire. What is the rationale for this selection? For example, the presence of Beagle suggests that recent provers are included, and the sentence does not intend to refer only to the three big ones (E, SPASS, Vampire). But then since there is Beagle, why not Zipperposition: Simon Cruanes. Extending superposition with integer arithmetic, structural induction, and beyond. PhD thesis, Ecole Polytechnique, Universite Paris-Saclay, September 2015.

We added the reference.

> Also, Prover9 is still in use, and since the authors cite it later, it should appear here. Actually, the authors cite a technical report on Prover9 dated 2006: does it really exist? The website https://www.cs.unm.edu/~mccune/mace4/ explicitly instructs to cite the web page for Prover9.

We added a citation of prover9 in the introduction, and changed the reference in the conclusion to a citation of prover9's website.

> The sentence "Nevertheless ..." with the reference to [22] of 2015 at the end of the first paragraph, and the following paragraph on SAT and SMT may give the impression that proof generation was considered first in SAT and only recently in ATP. One has to distinguish: proof compression may have been considered first in SAT and only lately in ATP, but proof generation in ATP is as old as ATP itself and much older than proof generation in SAT. Incomplete local search algorithms a la WalkSAT have been used for decades in SAT and obviously they do not generate proofs.

We now have:

"In contrast, although proof output has been a concern in first-order automated reasoning for a longer time than in propositional sat-solving, there has been much less work on simplifying first-order proofs."

Before we had:

"In contrast, there has been much less work on simplifying first-order proofs."

> DPLL is complete but does not generate proofs. Proof generation in SAT has become a standard only as a consequence of the inception of CDCL. Unlike DPLL, CDCL is guaranteed to generate a proof by resolution formed by the resolution inferences used to explain conflicts. This was observed for the first time in
Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications. Proceedings of the Conference on Design Automation and Test in Europe (DATE), IEEE, pages 10880-10885, 2003.
> As for SMT (before it was called SMT), proof generation was suggested as a requirement in
G. Necula and P. Lee. Efficient representation and validation of proofs. In Vaughan Pratt (Ed.) Proceedings of the 13th IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, pages 93-104, 1998.

Already in 1959/1960, in Davis and Putnam's paper "A Computing Procedure for Quantification Theory", in section 4 ("Feasible Methods in the Propositional Calculus"), one can easily see their rule III ("rule for eliminating atomic formulas") as the propositional resolution rule.

In 1962, Davis, Putnam, Logemann and Loveland (https://ia902606.us.archive.org/5/items/machineprogramfo00davi/machineprogramfo00davi.pdf), replaced rule III by rule III* (calling it the "splitting rule"). They remarked that "The forms of rule III are interchangeable".

Therefore, one could argue that DP is a method based on propositional resolution (although with a different name at that time), and therefore, capable of generating proofs at least in theory. Furthermore, one could argue that DPLL is a method capable of generating proofs, because one just has to replace rule III* (the semantic splitting) by rule III (resolution), as already observed by Davis, Putnam, Logemann and Loveland.

Robinson's resolution (1964) generalizes DP's rule III to first-order logic.

We prefer not to touch this historical controversy in our paper.

> If one considers that Otter had proof generation already at its first appearance in 1988, it is obvious that proof generation in ATP is Much older than proof generation in SAT or SMT.

We agree, and we had not claimed anything contrary to that in our paper. We just claimed that there have been more efforts to develop propositional proof compression algorithms than first-order proof compression algorithms.

In any case, any indirect false impression that propositional proof generation came before first-order proof generation in practice, should now be dismissed by our modified sentences, repeated below.

"In contrast, although proof output has been a concern in first-order automated reasoning for a longer time than in propositional sat-solving, there has been much less work on simplifying first-order proofs."

> Also theorem provers older than Otter, in the eighties, seventies, and sixties, generated Proofs. Since validity in first-order logic is only semi-decidable, the notion of a YES/NO answer does not make sense in first-order theorem proving,

That's a rather strong statement. From a practical perspective, a reasonably reliable "yes/no" answer, even without proof, is certainly better than nothing, in any logic.

And it doesn't even have to be 100% reliable. If you give me a tool that tells me, with probability 90%, whether famous mathematical conjectures hold, I would be willing to pay a positive amount of money for this tool. It would save me a lot of time, allowing me to focus my time on conjectures that are more likely to hold. Wouldn't you?

In fact, using not 100% reliable automated theorem provers as a "YES/NO" filter before trying to reliably re-prove conjectures within trusted proof assistants is a common approach (cf. Sledgehammer, HOLyHammer).

> and therefore the emphasis on proof generation has always been there.

Theoretically, it does not make sense to associate the need of proofs to (semi-)decidability.

The semi-decidability of validity in first-order logic only tells us that there are sound procedures that will eventually terminate and output "YES" when given any valid formula as input.

Imagine this:

1) You give me a formula (of which you don't know whether it is valid);
2) My first-order prover tells you "YES" without giving any proof;
3) You complain and say that you want a proof (senseless trying to justify your request for proof by saying that first-order logic is undecidable);
4) Then I tell you that I don't need to give you a proof, because first-order logic is semi-decidable and, therefore, you can implement any complete sound procedure of your own liking to check whether the formula is valid and whether my "YES" answer is correct.

So, if semi-decidability were to be used as an argument in a discussion about proofs, it would be in favour of **not** having to output proofs.

The most likely reason why you would want proofs becomes apparent in how the imagined situation above would unfold:

5) You would probably tell me something similar to this: "but I am lazy and I don't want to redo all your work; I am not sure I trust your YES answer (and I am not even sure I would trust the YES answer of my own procedure, if I were to implement it); I know it would be hard for you to show me that your procedure is sound, that its "YES" answers are always right; but couldn't you at least give me some hints or step-by-step instructions that would convince me that the answer for this particular formula is really YES?"

These are the main reasons why we need proofs: trust, complexity... not decidability. It is faster to check that a proof is correct than to reprove a theorem; it is easier to trust a small proof checker than to trust a large theorem proving procedure.

These reasons exist in propositional logic too. Checking a propositional refutation is in P, whereas showing unsatisfiability is in co-NP (cf. Cook).

> Partial regularization appears at the end of the second paragraph with references, but the text does not say what it is. It should.

The following sentence was added:

"[2] and [10] described a linear time proof compression algorithm based on partial regularization, which removes an inference η when it is redundant in the sense that its pivot literal already occurs as the pivot of another inference in every path from η to the root of the proof."

> The acronyms TSTP and GFOLU are not explained, not even expanded.

We have now expanded TPTP/TSTP at the first occurrence.

GFOLU was actually introduced only after its full expansion. No change.

> The Prolog-like notation that uses upper case for variables and lowercase for predicates may be a TPTP standard, but in journal articles it would be better to stick to the more classical notation with x, y, z ... for variables and P, Q, R, ... for predicates.

Changed as requested.

> Why use FV(t), FV(l), FV(Gamma), for the variables in a term, literal, clause? FV sounds like Free Variables, but the variables in clauses, and in literals and terms that occur in clauses, are not free, they are implicitly universally quantified. The standard notation is Var().

Syntactically, the variables are free. We attach an implicit universal meaning to them. Therefore, we consider it ok to use "FV". Nevertheless, since you prefer "Var", we have changed it.

Free variables are always implicitly quantified, and the quantification can be universal or existential, depending on conventions. Consider the following examples:

f(x) = x + 10 % x is a syntactically free variable, but implicitly universal

x^2 = 4 % x is a syntactically free variable, but now implicitly existential

> Also using Gamma for clauses is not standard. The standard notation is C, D, ... or maybe varphi, psi, ... although varphi, psi may be more in use for formulae or sentences. Gamma is typically used for something bigger, such as interpretations.

Gamma is frequently used to denote sets of formulas. For instance, in a sequent "Gamma \vdash Delta", Gamma and Delta are sets of formulas. Clauses can be seen as sets of formulas (literals). The standard of using Gamma, Delta, Theta and Lambda for collections of formulas goes back at least to Gentzen's 1935 paper ("Investigations into Logical Deduction"). It is a much older standard than using C and D, and it is still widely used today.

Moreover, using uppercase letters (P, Q, R and C, D) to denote objects of distinct kinds, such as predicates and clauses, as you suggested, is not ideal from an aesthetic perspective.

No change. We keep using Gamma.

> Also using "contraction" for factoring is not standard in theorem proving, and "contraction" already has other meanings in both theorem proving and proof theory. It is recommended to use factoring.

Factoring is just Gentzen's contraction with unification. Gentzen's contraction (which is already called "contraction" in his 1935 paper) precedes the other uses of the word "contraction" in theorem proving. Therefore, our use of "contraction" to refer to factoring is justified.

Nevertheless, to be coherent with our use of "resolution" instead of "cut (with unification)", we have renamed our contractions to factorings.

> Also, strictly speaking, resolution includes factoring. Thus, either we only say resolution, or else we say binary resolution and factoring. Since this article treats binary resolution and factoring as two distinct inference rules, they should be called binary resolution and factoring, respectively.

"binary resolution" is an unfortunate historical name, because it raises an implicature that "resolution" is not binary. We know that "binary" in "binary resolution" refers to the number of literals being resolved, and in this sense "resolution (including factoring)" is indeed not binary. However, words such as "unary" and "binary", when referring to inference rules, typically refer to the number of premises, and in this sense, it is misleading to raise an implicature that "resolution" is not binary. It is also odd that "binary resolution", despite being simpler and more atomic than "resolution (with factoring)", was given a more complex name.

We have added a footnote explaining that our "resolution" is called "binary resolution" elsewhere.

> Definition 2.1 is amazingly long, complicated, and hard to read for something as simple as proofs by first-order resolution. What will this formalism become if we were to extend it with contraction inference rules such as clausal simplification? What if we were to extend it to a system for first-order logic with equality including superposition, paramodulation, simplification, equational factoring? This is no way to give a proper, elegant inductive definition. In the literature there are already definitions of proofs by first-order resolution and beyond. They may not have all the decorations that the authors need, but enriching those definitions would still be better than this.

In the literature, it is typical to just show the schema of the resolution rule: two premises above the line, and the resolvent below the line; and then informally say that a resolution proof is a DAG of clauses derived with the resolution rule, and so on...

We need something more formal than that. We need a compact notation to describe proofs as first-class objects and the transformations that we can do on them.

With that in mind, definition 2.1 is actually a compact way of defining both the inference rules, the graph and all the notations and concepts that we need, simultaneously.

Nevertheless, to make it more digestible, we have split the definition in two. Now we have a definition introducing the inference rules in the traditional way before the definition of the proof graph and the proof notations.

> Several choices in this formalism for proofs are unexplained. For example, why do we need to store a mgu sigma as a pair of substitutions sigma_L and sigma_R?

That's a minor technical improvement. Suppose that we have the clauses: {P(x),Q(y)} and {-Q(c),R(x)}. With a single mgu sigma, we might carelessly (and wrongly) derive the resolvent {P(x),R(x)} (which is indeed a consequence of the premises, but is not the most general consequence). Informally, people usually avoid this problem by saying (or often silently assuming) that we must take variable-disjoint "variants" of the two premises, in order to correctly derive {P(x1),R(x2)}. This informality is insufficient in our case, because we must keep track of the instantiations of literals that occur in the proof (cf. the notion of instantiated resolved literal). We must store this renaming somewhere. And having two substitutions is a convenient, elegant and efficient way to do that: sigma_1 = {x\x1, y\c} and sigma_2 = {x\x2}. With a single substitution, we could not do that, because we would have to substitute x by both x1 and x2...

Note also that, the process of regularization involves deleting an edge, and it may be useful to know which "half" of the mgu is lost in this deletion.

> What does the circle with the central dot represent?

As stated in definition 2.1, it Phi_L \circledot Phi_R denotes a proof obtained by resolving the root of Phi_L and the root of Phi_R. You might ask: why didn't you write that, instead of using that heavy notation and have a graph-based definition? However, this notation makes it clear that nodes and edges may be shared among Phi_L and Phi_R. Furthermore, this notation makes it clear that the resolvent is uniquely determined with only the information explicitly stored in that notation.

Anyway, with the splitting of definition 2.1, this has hopefully become more digestible as well.

> Why do we need to state that the sets of variables have empty intersection,

Actually, thanks to the minor technical improvement I described above, we don't need to state that.
This statement is a remnant from a time when we were using a single substitution. We have removed it.

> when it is so much simpler to remind the reader at the beginning that each clause has its own variables?

See our comment above about taking variable-disjoint "variants" of clauses. Informal reminders like this are not sufficient in our case.

When we are doing theorem-proving, we may just rename variables and not care how they were called before. Therefore, we don't need to store the renaming precisely. But, in our case, we do need to do it.

> Assuming that there is no page limit or the page limit allows it, the material in the Appendix should be moved to the main text and explained properly.

Appendix moved to section 3; original section 3 removed.

> Line 3: "when" or "where"?

Fixed by replacement of section with appendix.

> Lines 5-9: "In the worst-case regular resolution proofs ...": "regular resolution proof" is not defined; "exponentially bigger" in which measure?

Fixed by replacement of section with appendix.

> Clearly, this section summarizes notions that were defined elsewhere. However, a journal article should be self-contained and key definitions should be reproduced formally.

There is a page limit, and we are very close to reaching it. Therefore, there is a limit on how self-contained this paper can be.

> The notion of "safe" literal should be defined.

By moving the appendix into the main matter of the paper, the paper is now more self-contained and the notion of safe literal is now clear even to readers who had not read papers about the propositional version of the algorithm.

> This section needs an example badly.

Added an example.

> What is the RP algorithm?

This question is answered in the introduction.

No change.

> As for the Appendix, the notion of proof context and its notation are used without having been introduced.
Already after the second paragraph an example is needed.

Proof context has been defined, and a small example has been added.

> The title only says "First-Order Challenges": is any of them solved in this paper? As it stands, we go from "challenges" to "implementation", if one goes over the paper reading only section titles. It sounds weird.

Changed to “Lifting to First-Order”.

> Example 4.1 is not clear because the propositional notion of safety was not defined.

By moving the appendix into the main matter of the paper, the paper is now more self-contained and the notion of safe literal is now clear even to readers who had not read papers about the propositional version of the algorithm.

> Example 4.2: is there a reason for considering eta_3?

A naive attempt to generalize safe literals to the first-order case would consider eta_3 as the unification of the safe literal and the resolved literal is possible. However, doing so prevents the conclusion of the proof. This example is intended to illustrate the need for greater care; we have reworded it to make this more apparent.

> It may help to use the subsumption ordering in order to capture the notion of being more general.

Changed - subsumption defined in Section 2 and used throughout the paper instead of “more general”.

> Line 16: "modification to" --> "modification of"

Changed.

> What is the intuition, the meaning of being "pre-regularizable"? The text that follows Definition 4.2 is a restatement that does not explain.

A sentence added in the new version explains that this is, intuitively, a *necessary* condition.

> Example 4.3: what neg q(R,S) and not q(T,V)?

Assuming that you mean “why neg q(R,S) and not q(T,V)?”, the response is that *instantiated* safe literals are stored. The negation is maintained, since we store literals (not atoms), and the resolution uses \sigma_L={T->R, V->S}. No change. However, we did change some definitions, making it clear that *instantiated* safe literals are stored.

> Definition 4.4: how can we have S(eta_1) before having S(eta)?

We don't understand this question.

> What is the meaning of l = l_ssigma_s in S(eta_1)? Once sigma_s is applied, we only have l in the set S(eta_1). Does this mean that we should store l as (l_s, sigma_s) in S(eta_1), that is, without applying the substitution?

Added a comment to suggest that this is one way the collection of safe literals could be implemented for this condition.

> Also, the notion of "corresponding" is undefined.

We avoided the word "corresponding" and reworded the sentence as follows:

"[...] where $\dual{\ell}^{\dagger}$ is the literal in $\dn{\psi}{\eta_2}$ that used to be\footnote{Because of the removal of η_2, $\dual{\ell}^{\dagger}$ may differ from $\dual{\ell}$.} $\dual{\ell}$ in ψ [...]"

> Example 4.4: all mgu's are variable renamings: is this kind of situation simpler? A comment is required. The readability of this example would improve if the bottom up computations of the S() and R() sets were shown.

Added a table to show computations of the sets S() and R(). Changed example slightly so that it no longer consists of only renamings - we don’t want to imply that an all variable renaming situation is why weak regularization works.

> Why are the literals in S(eta_8) negated? It does not seem that sign plays a role. If this is true, then why not use atoms? This example is not understandable.

Sign does play a role. Safe literals are literals (i.e. atoms or negated atoms).

No change.

> "With logical soundness guaranteed ...": really? The soundness of implementations is not trivial.

That sentence was just an unnecessary summary of the following sentences that preceded and supported it:

"Note that by implementing the algorithms in this library, we are able to guarantee the correctness of the compressed proofs, as in Skeptik every inference rule (e.g. resolution, factoring) is implemented as a small class (at most 178 lines of code) with a constructor that checks whether the conditions for the application of the rule are met, thereby preventing the creation of objects representing incorrect proof nodes (i.e. unsound inferences)."

We have removed the unnecessary summary sentence and improved the preceding sentences to:

"Note that by implementing the algorithms in this library, we have a relative guarantee that the compressed proofs are correct, as in Skeptik every inference rule (e.g. resolution, factoring) is implemented as a small class (each at most 178 lines of code that is assumed correct) with a constructor that checks whether the conditions for the application of the rule are met, thereby preventing the creation of objects representing incorrect proof nodes (i.e. unsound inferences)."

> "the advanced inference rules of SPASS were disabled": which ones?

We have now listed the only two enabled inference rules: "Standard Resolution" and "Condensation".

> The paragraphs in this section are too long.

We have broken the paragraphs and added subsections to group paragraphs with a common topic.

> " ... it is not a priori clear ...": how can it be "a priori" since it is after the experiments?

Obviously, for each proof used in the experiments, we know which combination performs better. But for a **new** proof, it is not a priori clear which combination would perform better. The only known way to find out is to run both. Before running both (i.e. **a priori**), we do not know.

We have changed that sentence to:

"Therefore, as in the propositional case [10], it is not a priori clear which combination will compress a proof more."

> In the last paragraph, what is the point of comparing the time it takes to find a proof with the time it takes to compress it? It sounds like comparing apples and oranges.

Oranges and apples are not so different from each other. They are both fruits. If we had to tell you the nutritional content of an apple, it would make sense to compare it with the nutritional content of oranges and other fruits. This would be relevant for you, as it would allow you to decide whether to combine apples and oranges in your diet.

Likewise, proof search, proof checking and proof compression are related tasks, and it makes sense to compare the time they take. For instance, people normally expect proof checking to be faster than proof search (otherwise, why care about checking somebody else's proof? We could just search for our own proof instead). Depending on the application context, there are similar expectations about proof compression time.

No change.

> It does not make sense to cite Kowalski-Hayes (1969), Maslov (1964), and then Waldmann (2017) for ordered resolution. Ordered resolution in the current state of the art refers to a restriction of resolution where we resolve only on maximal instances of literals, with respect to an ordering that is at least a reduction ordering (stable, monotone, well-founded) and preferably a complete simplification ordering (stable, monotone, with the subterm property,
and total on ground). None of this existed at the time of Kowalski-Hayes and Maslov. A proper starting point is
Jieh Hsiang and Michael Rusinowitch. Proving refutational Completeness of theorem proving strategies: the transfinite semantic tree method. J. ACM 38(3):559-587, 1991. which also contains a section on the pre-existing notions.

Removed references to Kowalski-Hayes and Maslov; kept reference to Waldmann; added reference to Hsiang and Rusinowitch.

> It does not make sense to cite only Overbeek (1975) for hyperresolution. Hyperresolution was invented by J. Alan Robinson:
> John Alan Robinson. Automatic deduction with hyper-resolution. International Journal of Computer Mathematics, 1:227-234, 1965.

Reference to Robinson added.

> The authors state that "Resolution restrictions and refinements Tend to result in longer chains of resolutions": the above refinements were conceived mostly to save space with respect to plain resolution, by doing fewer inferences and generating fewer clauses. Thus, the authors suggest that these refinements may allow us to gain space and time during the search for the proof but at the expense of generating longer proofs?

Yes. That is (almost) what we mean. We would add a "possibly" there: "at the expense of possibly generating longer proofs".

> How do we know that this is true? Is this a formal result? Where did it appear? Or is it a claim by the authors? How supported?

We believe it is a folklore intuition. Depending on how we state it, it is also an easy result. Let C be a clause set, and let R(C) be the set of all unrestricted resolution refutations of C. Let P be the shortest refutation in R(C). Now let R'(C) be the set of resolution refutations of C that satisfy a given refinement/restriction. Clearly R'(C) \subseteq R(C). Therefore, it is *possible* that P is not in R(C). Let P' be the shortest proof in R'(C). It is possible that P' is longer than P. Intuitively, the more restricted a refinement is, the more proofs it will rule out, and thus the higher the likelihood that the shortest proofs in R(C) will not be in R'(C).

I remember seeing (in some book) a concrete example where this happened for lock resolution. Consider also the level saturation strategy that you mentioned. Since it generates only fully regular proofs, there will be clause sets for which the shortest level-saturation proofs are exponentially longer than the shortest unrestricted resolution proofs.

We have changed "tend to result" to "may result", and we added the following sentence in the introduction:

"[...] and the most efficient provers do not necessarily generate the shortest proofs. One reason for this is that efficient resolution provers use refinements that restrict the application of inference rules. Although fewer clauses are generated and the search space is reduced, refinements may exclude short proofs whose inferences do not satisfy the restriction."

> Further down, the authors write "It is conceptually easy to Adapt the algorithm described here to such variations of resolution.": maybe it is true, but how do we know? What is the evidence, the support for this claim?

The rest of that paragraph supports the claim. The next two sentences discuss why restrictions/refinements are trivial (namely, because every restricted resolution proof is also a resolution proof). And the last two sentences of that paragraph briefly discuss two extensions (equality reasoning and splitting).

Nevertheless, we have weakened the claim to "It is conceptually easy to adapt the algorithm described here to many variations of resolution".

> The year is missing in reference [5].

The year has been added to the reference.