Dear Editors,

We would like to thank the referees for the thorough reviews that they provided us with. Their comments proved valuable for improving our paper. In the revised version that we are submitting, we have changed the title and the structure of the paper to better separate the grammar description from the exploratory study of the datasets, which had been strengthened to explore specific research questions. Also, this version has an updated grammar description and diagrams to match the implementation, more detailed description of the dataset, updated grammar requirements and more detailed explanation for the selected parser generator.

Below we respond to the reviewers’ comments and we explain how we revised the paper to address them.

Looking forward to your feedback on the revised version of the paper.

Comment 1.1 *Although the use case is notable (spreadsheet formulas) the described language itself is not. We are supposed to know how to create grammars for expression languages. If this is not the case for the current language, then you have to convince what theory or technology is lacking to construct the current parser. Is there a problem you are solving with this parser? What is the novelty? The paper does not convince me of a technical or theoretical issue of significant interest to warrant research.*

Answer 1.1.Our intention with this paper it to contribute to the spreadsheet research community, not the language engineering one. The problem that we are addressing with the parser is that there are numerous works in the spreadsheet research field that analyze spreadsheet formulas using either simple grammars which have not been evaluated and often contain obvious errors, or using implied, undefined grammars. We believe that a reliable grammar and its parser implementation, available to the spreadsheet research community, can support spreadsheet research and can enhance the understanding and usability of research results. Our motivation for publishing the parser is better explained in the second and third paragraphs of the introduction section of the paper.

Comment 1.2 *The claim that this grammar/parser is "bulletproof" is both vague and not substantiated. Parsing a corpus can be done by any grammar which over-approximates the structure and words of a language. If a parser can parse a corpus without crashing or producing parse errors, then you have done a first engineering step. Well done, but so what for the advance of software engineering? The next engineering questions are if the parser is precise (does it reject sentences which are not in the language?), if it is correct (does the structure reflect the semantics of the language) and is its definition unambiguous (does it uniquely define the structure of the sentences?). Neither of these questions are addressed by the paper with sufficient rigor, not conceptually, not in the experimental method, and thus not in the results. A contribution might be if (new) methods for answering these questions would have been explored and validated or compared to existing methods.*

Answer 1.2We agree that the bulletproof claim is too bold, and we removed it in this version. Also, in the requirements that we set for the parser we added one against over-approximation, that it must recognize the spreadsheet formula elements that are required for supporting spreadsheet research. The rejection of invalid formulas and the correctness of the parse trees are discussed in sections 6.3 and 6.4. We consider it, however, out of the scope of this work to further explore, validate and compare methods for answering those questions.

*Comment 1.3 If we take the real contribution of this paper, the synthesis of the report on the use of the language in the corpus, then this study is lacking in the expected rigor of a field study. What are we studying? Why? How? What did we want to learn and did we actually learn? None of the normal reflection on the academic level is present in the paper.*

Answer 1.3 Thank you for pointing this out. The analysis of the datasets was “buried” in the previous version, under the scope of the evaluation of the parser. In this version, we utilize the parser to conduct an exploratory study of the formulas in the datasets. We formulate specific research questions, which we address in a separate results section. We also updated the related work to include similar studies.

*Comment 1.4 The current paper is an extended version of the SCAM 2015 paper titled "A Grammar for Spreadsheet Formulas Evaluated on Two Large Datasets". I did not see a significant delta with respect to this paper.*

Answer 1.4 Our understanding is that this submission is supposed to be an extended version of the SCAM paper. We consider the delta to be significant, even more in this revised version. In comparison to the SCAM paper, considering only the most significant changes, the evaluation is based on an eight times bigger dataset, the grammar and the analysis are updated to address the new constructs found in the more spreadsheets, the design process is explained in a new section, a structured exploratory study of the datasets is presented, the most common functions, operators and constants are identified, and complexity characteristics of the formulas are studied.

*Comment 2.1 There are some issues with the formula grammar proposed in this paper being bulletproof. First, as we discover around page 11, a couple of bullets do get through and some of them are impossible to dodge either. More drastic than that is the fact that the entire work is focused ultimately and exclusively on Excel. Since we do not call papers that propose good grammars for, say, Java, “Bulletproof Grammars for Programming Languages”, it would be equally inappropriate to rob this paper of having “Excel” somewhere in the title. The issue is discussed in extreme brevity and without any technical substance in section 5.1, and this position has not been strengthened in any way since the conference version of the paper. To make a steadier bulletproof case for general coverage of spreadsheets, one would need to have at least some superficial research done on Calc, Numbers, Gnumeric, Pyspread, Calligra, etc. For instance, at least Calc and Gnumeric work seamlessly with .xlsx files. Can they read all 100k files? Are there any sizeable collections of files created natively by them? How does the list of Excel functions covered by the XLParser relate to the Google Spreadsheets function list by Magnus Adielsson et al? What stopped the authors from covering the tilde operator from Calc into their grammar?   
  
There are two ways to approach this problem.   
- If the authors opt for putting Excel in the title, the contribution becomes more clearly defined and articulated, and little needs to be changed in the paper (modulo other comments below).   
- If the authors opt for working on a general case, many sections of the paper need to be rewritten, since they continually refer to .xlsx extensions, to specific keystrokes, to tables being an equivalent term to sheets, etc, throughout the paper.*

Answer 2.1 We agree that the title of the paper was too bold, especially since only Excel formulas are considered in our study. Thank you for the suggestions on the pathway towards extending the grammar to the general case. For this work, we prefer to keep its scope to Excel formulas, so we updated the title to a more modest one, specifying that it concerns Excel formulas only.  
  
*Comment 2.2 Spreadsheets are very important, certainly omnipresent and undoubtedly widespread. Still, it is suboptimal to rely on the Bradley-McDaid claim that “90% of desktops have Excel installed” right in the first paragraph of this work. They do not “estimate” it, they handwave the high percentage in the abstract of their poster paper and never get back to it either to elaborate nor to attribute it to anyone’s research. This estimation also badly corresponds to the data that at most 85% of desktops run any version of Windows (this is the most optimistic estimation by StatCounter, there are many different ones going as unbelievably low as 50%), which, combined with Excel’s inability to run on any kind of Linux/Unix, would roughly mean that it has to be found on each Windows computer as well as on half the Macs.*

Answer 2.2 Thank you for pointing this out. We removed this reference and added more reliable sources.  
  
*Comment 2.3 Which version of Excel is used? It is clear that the authors could not identify that clearly with such a large scale empirical study, but it would be interesting to get some estimation — Excel is known to change their language with every release, introducing new features and functions. In particular, it is valuable to know whether these changes are irrelevant to the formula grammar at all (e.g., they add new units to CONVERT each time, but the way the paper authors universally treat arguments of any function, provides a universal solution) or still relevant to some extent (e.g., a structured hash-reference briefly talked about on page 17). As it stands now, the readers do not even know if Gembox that you use to convert \*.xls to \*.xlsx, introduces any changes concerning function names or performs any other adjustments beyond the one that is described on pages 10-11.*

Answer 2.3 We agree, these points were not sufficiently covered in the paper. The original spreadsheets in the datasets are of various Excel versions. The Euses and the Enron datasets contain spreadsheets in the Excel binary file format, while the the Fuse and the Web datasets include more recent spreadsheets, saved in Excel 2007 and later workbook format. We added an explanation for this in the datasets description in the fourth paragraph of section 4. Also, we enriched Section 6.1 to explain that the Gembox conversion does not affect the function names of previous Excel versions that have been deprecated after Excel 2007, but it does adjust deprecated syntactical features like regular expressions and natural language formulas. Also, we added a paragraph (second in 6.1) on the changes that would be relevant or irrelevant to the formula grammar.  
  
*Comment 2.4 In section 3, the authors present three requirements that they want the formula grammar to fulfil to be “bulletproof”. The requirements seem legitimate yet #3 is not. First of all, “be compact \*enough\* to \*feasibly\* implement” is a super vague statement based on two imprecise, unmeasurable parameters. Second, parser generators are used mostly for the reason of overcoming this obstacle and reliably generating code based on \*any\* number of grammar elements, so having hundreds of production rules is not a show stopper — on the contrary, it is a very good reason to use it. Thus, disregarding official formula grammars on the basis of their size is wrong and misleading for the readers: in reality you have probably disregarded them for reasons of quality. There is nothing fundamentally wrong about it, and grammar recovery, adaptation and debugging are far from trivial and/or industrialised, so a decision to develop a new grammar is sensible, especially if supported by a large testing endeavour, but it has to be honestly explained. The second reason given in the same introductory paragraph to section 3 is closer to the substance, but again, if the entire problem was solvable by inlining a bunch of superficial nonterminals, the authors would have done that with any grammar manipulation framework of their choice.*

Answer 2.4 We agree, requirement #3 was both too vague and misleading and we removed it from the paper. The official grammar is essentially disregarded because of its different purpose. We wanted a grammar that is, for our purpose of doing spreadsheets research, “as simple as possible, but not simpler than that”, a point that we made in our presentation in SCAM but we had not explained sufficiently in the paper. We updated section 3 to explain this. We also added a requirement that we believe must be made explicit, that the grammar should recognize the spreadsheet formula elements that are required for supporting spreadsheets research (a requirement against oversimplification).  
  
*Comment 2.5 The use of Irony as opposed to ANTLR or any other framework, is not argumented, even though closely related issues like the grammar class are discussed (e.g., in section 3.2). The way one would expect this issue to be handled in the paper: we needed features X and Y, there are tools A and B that fit, this is a quick comparison between them, long story short, we opted for A because of these reasons. The way this issue is tacked in the current version of the paper: we want human understandable parser, so we choose LR parsing which is known to be more convoluted and harder to debug than LL; our grammar does not belong to the LALR class, so we choose an LALR parser generator; however, we need extra features to deal with ambiguities, but they are present in all LALR frameworks, so we choose Irony.*

Answer 2.5 We added an explanation in the second paragraph of Section 4 about the reasons for choosing Irony: It supports the required features (precedence, manual resolution of shift-reduce conflicts, implicit operators) while offering a good integration to the .NET platform that we also used for implementing the evaluation tools.

*Comment 2.6 One does not simply misspell “xlsx” in a paper about Excel spreadsheets, as it is done in a code fragment on page 5, line 28.*

Answer 2.6 Corrected in this version.  
  
*Comment 2.7 Grammars: even though Irony grammars are not famous for their readability, it is somewhat unfortunate that neither the grammar fragments nor the syntax diagrams found in the paper, were derived from the actual artefact. The manual process of their conversion (which had to be manually tested) led to the introduction of the following problems:   
- There have been some decorative refactorings that make it harder to check the three for consistency (e.g., TEXT vs STRING, many folded/unfolded nonterminals, repositioned brackets, etc).*

Answer 2.7 Manual conversion is indeed error prone. We believe it is worth it, though. The ebnf specification is much more readable than the C# one, and most readers will be more familiar with it. In this version we corrected all issues that you mention (thank you for checking and spotting them), we also double checked it, and it matches the implementation.

*Comment 2.8 Many places in the illustrative grammar are still hard to decipher: Table I plays the instrumental role there, explaining that NR is a named range and DDE is a dynamic data exchange, as well as the totally unexpected fact that the opening bracket is included as a part of the Excel function name and a reference function. Is there a way to make it more readable? Decoupling it from the grammar and printing on a separate page in rotated form does not encourage reading and cross-checking.*

Answer 2.8 We tried different options. We cannot make it fit if not rotated while keeping the regular expressions in single lines. We removed the table and moved the definition of the lexical tokens in a separate section, but using regular expressions in the text looked worse. Also, we found that it made it harder to search for token definitions. For this reason, we left it as it is in a separate table. We clarified in the token description of functions that they include the opening bracket.

*Comments 2.9 & 2.10 The start symbol from a grammar in its first alternative allows any Formula in the implementation yet only Constants in the paper. The Prefix nonterminal seems to miss two alternatives present in the implementation: the quoted multiple sheets and the quoted file multiple sheets.*

*Answers 2.9 & 2.10* True, we modified the terminals and non-terminals descriptions to fix both issues

*Comment 2.11 The REF-FUNCTION-COND in (called ConditionalRefFunctionName in the implementation) is present in the paper on the diagrams, moved around but also present in the grammar; yet it is commented out in the artefact with the functions themselves (that are listed in the footnote on page 13) merged into the main list. What is the reason for these differences?*

Answer 2.11 The REF-FUNCTION-COND in the paper (called ExcelConditionalRefFunctionToken in the implemntation) is used in the <RefFunctionName> in the paper (also RefFunctionName in the implemntation). There are no differences between the implementation and the representation of those rules in the paper. The commented out ConditionalRefFunctionName was leftover code from a previous implementation version, which we cleaned up in the current version of the implementation. The list in the footnote in page 13 is of the conditional functions, not of the reference-returning functions. This was indeed confusing, so we added an explanation for the reference-returning functions in the second paragraph following the grammar rules.

*Comment 2.12 The prefixed named range described in the paper, seems to be missing from the implementation.*

Answer 2.12 It is named differently. The equivalent of the NR-PREFIXED token is the NamedRangeCombinationToken in the implementation. We renamed NR-PREFIXED to NR-COMBINATION in the paper.

*Comment 2.13 The StructuredReference given in the paper is way more permissive than the StructureReference (NB the unreasonable inequality in the names!) in the implementation.*

Answer 2.13 We agree, the inequality in names is confusing, and the structured references grammar description was not the same as the implementation. We renamed the tokens in the implementation to match the ones in the paper, and we updated the grammar rules for the structured references in the paper to match the implementation.

*Comment 2.14 The syntax diagram of Formula contains an erroneous overapproximative attempt to combine prefixed and postfixed formulas.*

Answer 2.14 Updated the formula syntax diagram to split the prefixed and postfixed formula branches.  
  
*Comment 2.15 In section 4.1, there is an unexpectedly painful moment when Gembox shortcomings are discussed: it seems like the process involved manually copy-pasting 371 formulas one by one and fixing them, effectively introducing a human preprocessor. If these cases are explained correctly, then this pattern was quite possible to automate and either provide a normaliser that would fit in a tool pipeline after Gembox and before XLParser, or implement a fallback. Either of those ways would have been less work, and still perfectly testable (even manually, if you must), and then serve both as a means of completing the experiment and as solid evidence that the entity-with-spaces-without-quotes pattern indeed covers all problematic formulas.* 

Answer 2.15 We agree, by automating this process we would have saved time and we have produced more testable results for the failing formulas. However, the problems introduced by Gembox are not predictable, they change with each version. Before the SCAM submission, we had notified them about the mismatched formulas that we had found and in the current Gembox version most of those types of errors are indeed fixed. We have notified them about the two types of errors that remain (the last two cases in Section 4.1), and we expect them to be fixed in the next version. We did not automate this process because we hoped that this problem would be a temporary one. Instead, we used a flat file representation for fixing the incorrectly read formulas and using them as input to the parser. We added this note about our intermediate representation in Section 4.1.

*Comment 2.16 In section 4.2 some plots are provided obtained with metrics that are introduced in running text in a seemingly ad hoc manner. Was there a reason for that? There are already proposed, researched and evaluated suites for metrics on grammars, trees, graphs, etc, please consider using those or elaborating why none of those apply to your particular dataset.*

Answer 2.16 The title (“Parse trees”) and the positioning of this section was misleading: The metrics that were presented were not metrics on the grammar or on the parse trees, but metrics related to the complexity of the formulas in the datasets –conditional depth, for example, essentially represents the nestedness of the conditional operations in the formulas. We calculated these indicators in order to demonstrate how the produced parse trees can be used for analyzing formula characteristics. In this version we have removed this section, moving part of its content to section 5.1 which explores formula complexity indicators.

*Comment 2.17 In section 4.3 about grammar analysis some statements become ambiguous because of the word “formula” which might be a formula as in “a formula grammar” — a top level formula, so to speak, or it might be a formula as something parseable with the Formula nonterminal. Are only top formulas included when you say that 86,61% of them include a function call? Table IV seems to use the second meaning of a formula.*

Answer 2.17 Yes, the meaning is that 86,61% of the parsed formulas in the dataset contain at least one <FunctionCall>. Table IV uses the first meaning, it shows the number of parsed formulas in the dataset that contain each type of node. 100% of the parsed formulas will contain a <Formula > node. We updated Section 5.2 with this explanation.  
  
*Comment 2.18* On page 8, line 50 says: “ and are the two most important production rules”. Technically speaking, and are nonterminal symbols that designate production rules.   
*Comment 2.19 Page 11, line 20: “comprising of” => either “consisting of” or “comprising”*

Answer 2.18 & 2.19 Corrected both in the revised paper.