Reviewing: 1   
  
Comments to the Author   
I have read this paper with interest to find it a well structured, readable and understandable report on the construction of a parser for spreadsheet formulas. The parser is used to parse a large corpus of formulas and the resulting parse trees are also used to statistically observe properties of these trees. In this sense the paper synthesises an overview of this corpus in terms of the kind of formulas which are used in it.   
  
I appreciate that the construction and testing of this parser was hard work, and that it is a necessary step in doing more research in the analysis of spreadsheets.   
  
I have selected a reject for this paper for the following reasons:   
\* 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.   
\* 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.   
\* 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.   
\* 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.   
  
That's a pretty harsh conclusion, I admit, and I do hope it does not deter you from continuing your work in analysing spreadsheets, but this intermediate step in your journey is just not publishable in my opinion.   
  
Reviewing: 2   
  
Comments to the Author   
This paper addresses an important problems in spreadsheet research, describing development and evaluation of a grammar for formulas found in Excel spreadsheets. It is an extended version of the SCAM publication with a similar yet slightly more modest name. The extensions include slightly different layout, slightly more elaborate explanations, and the extension of the test data set eightfold. In general it has the “more of the same” feel, but such a drastic test fodder expansion must have eliminated some of the harder to find bugs in the grammar compared to the SCAM’15 version, and thus this reviewer believes that this paper deserves a chance at JSME.   
  
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.   
  
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.   
  
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.   
  
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.   
  
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.   
  
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.

Corrected both in the revised paper.  
  
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).   
- 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.   
*- 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.*

True, modified the terminals and non-terminals descriptions to fix both issues  
*- 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?*

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 is leftover code from a previous implementation version, which we plan to clean up in the near future. 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.  
- The prefixed named range described in the paper, seems to be missing from the implementation.   
- The StructuredReference given in the paper is way more permissive than the StructureReference (NB the unreasonable inequality in the names!) in the implementation.   
- The syntax diagram of Formula contains an erroneous overapproximative attempt to combine prefixed and postfixed formulas.   
  
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.   
  
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.   
  
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.   
  
On page 8, line 50 says: “ and are the two most important production rules”. Technically speaking, and are nonterminal symbols that designate production rules.   
  
*Page 11, line 20: “comprising of” => either “consisting of” or “comprising”*

Corrected both in the revised paper.