Reviewer 1

Next, we provide answers (in blue text) to the comments/remarks pointed out by the reviewers (in black text).

Thanks for the careful reading and for the comments and suggestions. Below we describe how we dealt with them.

The paper presents SPLAP. It is a probabilistic extension of SPLA, a process algebra for specifying software product lines (SPL) presented in previous works by the authors

Details about each section

Section 1 Introduces the paper and mentions related approaches. But no comparison with other approaches is actually given.

In this version of the paper we have included a new section (see Section 2. Related work), which contains a comparison with current work that can be found in the literature.

Section 2.1 Introduces the syntax and operational semantics of SPLAP. The authors also show that the operational semantics of SPLAP agrees with the one of SPLA, if probabilities are discarded from the derivations. I believe that the description of the syntax delegates too much to previous works of the authors presenting the syntax of SPLA. More intuition should be given about the role and meaning of each operator. For example:

- How should conjunction be interpreted?

P $\vee$ Q is the conjunction, intuitively it combines the products of both subterms P and Q by accumulating the features. There is an example (Example 2) in the paper to illustrate this.

- The operator 'P1 choose-one_p P2' allows choosing probabilistically among the two SPL P1 and P2 with probability p and 1-p, respectively. Then, it is said that this operator is actually n-ary. How should be interpreted such n-ary version? Which are the probabilities involved?

We have added more comments after Definition 1 and Examples 1 and 2 and 3 to explain the syntax. We have removed the comments about the choice being n-ary. We could have defined n-ary (by assigning a probability distribution to the n options), but we think the notation would have been more complex to follow.

Section 2.2 Introduces a denotation semantics for SPLAP, while **Section 3** proves that the two semantics are equivalent.

Section 4 Extends the language with a new operator to 'hide' in the semantics part of the features we are not interested in. This allows to obtain more compact state spaces, and hence to handle larger models.

I am not sure about this section for two reasons:

- I am not sure about the relevance/importance of this section in the paper. The presented notion of hiding is not considered/mentioned in any other section.

The hide operator is important for the implementation. It maybe not possible to compute all the SPL, it may be feasible to compute the probability of a particular feature. It is achieved by hiding sets of features, that is, those “hidden” features do not affect the processing of the probability of a feature X for being part of a valid product. We mention this in the new version of the Introduction and we have included a paragraph with an example in the Empirical Study Section to explain it.

- Why do you need a new section for this operator? Couldn't you present it in section 2?

The hide operator is not a usual one in SPLs. So, we believe that a dedicated section is adequate in order to present the hiding operator.

Section 5 Validates the approach by running its tool instantiation against large SPL models. First, it considers a model with 1500 features generated using the tool BeTTy. It shows how to study the probability of having each of the 1500 features (and the runtime of this analysis). Then, the authors perform a scalability analysis of their tool for models with up to 10000 features. Three classes of models are considered, with different configurations in terms of presence of optional/mandatory features and choose-one/conjunction operators. I have two concerns about these analyses:

First, in the second-last paragraph of page 12, the authors justify unexpected variations in the runtimes in terms of disk latencies, memory paging, etc. These problems can be solved by performing each experiment many times (e.g. 10), and providing the average runtime for each experiment. In page 13 the authors mention that they repeated experiments more times, but apparently they chose one runtime.

The experiment described in Section 6.1 Model Analysis, has been repeated 10 times. A table containing the obtained results has been included in this section (see Table 1). Also, we clarify that the obtained results might not be only caused by the latencies (disk, memories, operating systems, etc..) but the stochastic nature of the generated models also has an important influence on the tendency of these results.

Additionally, for each performance experiment described in section 6.2, each feature model has been generated 30 times. The results are presented in a box-plot chart, to show the execution time for processing these models, and a line chart, for representing the average consumption of memory.

Second, I cannot understand how the models have been generated using BeTTy. In page 12 the authors say:

- the probability of having a mandatory feature is 0.2,The proof of the following result is an immediate consequence of Lemmas~\ref{lem:pref}--\ref{lem:excl} (see Appendix~\ref{app:proofs} of the paper)
- … having an optional feature is 0.3,
- … having a feature in a choose-one is 0.25,
- … having a feature in a choose-one is 0.25.

To the best of my knowledge, a mandatory feature is a feature that must be present in every product generated by the product line. Hence, I would have expected to find that 20% of the features have probability 1 of being installed. However, as shown in figures 5 and 6, this is not the case. I believe that the authors should give more intuition about the nature of the models they consider.

This is a good point. Since this concept was not clearly described in the first version of the paper, we have re-written this part to clarify the values for configuring BeTTy and the obtained results.

In this case, it is important to differentiate the probabilities defined in BeTTy, which are used to generate a feature model, and the probability - calculated from the model - to have a feature in a final product. Let suppose that we configure BeTTy to generate a feature model using a probability of 0.2 for having a mandatory feature, which means that 20% of the generated features are mandatory. However, this does not imply that these features are part of the 20% of the generated products, because the probability of having a feature in a final product depends of the position of this feature in the model. For example, if a given mandatory feature is placed in a choose-one relationship, it is possible that the other branch (the branch that does not contain the mandatory feature) is used to generate the final product, discarding the branch containing the mandatory feature.

Discussion
In addition to the observations done above, I believe that the authors should improve the accessibility of their work. In particular, sections 2, 3, and 4 are long and contain many definitions and results. The authors should provide more motivation and examples. I would suggest the authors to use a running example starting from section 1 or 2, and use it to better explain/exemplify parts of SPLAP.

In the new version of the paper a new example has been added (see Section 3) to clarify how the general semantics works.

Furthermore, the authors show that the implementation of the denotational semantics of SPLAP is much more time/memory efficient (many orders of magnitude) with respect to the one of SPLA. But no explanation for this is given. In the first 3 lines of the conclusions, the authors say that the motivation behind the proposed probabilistic extension is to alleviate the state space explosion problem. Why is that? Why should the probabilistic extension alleviate such problem?

In this work, we refer as alleviating the state space explosion problem in the sense that large models can be efficiently processed in a reasonable time frame. In a previous experiment, our implementation of the denotational semantics from SPLA required around 300 seconds to process a model containing 17 features. However, in this paper we show that, using our implementation of the probabilistic extension, we are able to process – in less than 300 seconds - models containing 13.000 features.

Finally, given that this is a journal paper, I am surprised by the fact that proofs are provided in two appendices. Will these appendices be published? Will they be made available online?

We put the proofs in the appendix just for the sake of readability and we expect appendices to be published as well.

For these points, and for the ones raised above, I suggest the authors revise their submission.

Minor comments:

- Please rephrase the sentence 'The results are promising .... (450 featurThe proof of the following result is an immediate consequence of Lemmas~\ref{lem:pref}--\ref{lem:excl} (see Appendix~\ref{app:proofs} of the paper)es).' I can't understand what it means.

This has been fixed in the new version on the paper.

- Among the related works, you did not mention ProFeat, a probabilistic language for SPL based on PRISM (https://link.springer.com/article/10.1007/s00165-017-0432-4)

This article is fairly recent and it has been cited in the new version of the paper.

- You mention three tools: QFLan, Z3, and Multivesta. I would suggest to provide a citation for each.

The new version of the paper includes these citations.

- Why do you forbid probability values 0 and 1?

These values give degenerate terms that are covered by other operators. We do not consider probability 0 in the optional feature because it would be equivalent to nil. We do not consider probability 1 in this operator because it would be equivalent to the ordinary prefix operator. It is true that we could have merged both operators, but we have preferred to follow the tradition of having both kinds of features: mandatory and optional. In the case of the choice operator, these values would give degenerated terms in which one of the sub-terms is never considered.