OOPSLA 2022 ROUND 2 RESUBMISSION CHANGES

We detail here what changes we have made to our round 1 submission after feedback and advice by the reviewers. We wish to thank the reviewers. The reviewer suggestions have been very helpful, and assisted us in identifying aspects of our paper that needed improvement. We have made significant changes to the paper and its structure to a large degree due to the input of the reviewers.

Firstly, the reviewers requested the following changes be made:

1. Mandatory Changes

1: *Clarify novelty/increment over existing work. In particular (but not only) over Chainmail*

- We have rewritten the introduction to better place in the related work (lines 12-19), and more explicitly address the novelty of our contribution (lines 55-60), and how it differs from existing systems such as Chainmail and VerX (lines 60-64). In the related work section, we compare with VerX/Chainmail in more depth than before (lines 1163-1174). In addition, we compare with temporal loig (lines 5450554)

2: *Vastly expand on explanation/intuition of the $\triangleleft$-operator. The explanation from the author response is not at all convincing nor intuitive. It is yet too terse. Both an example and/or a picture would also help greatly. It would also help greatly if later when the operator is used in other definitions, it was explained just exactly how this operator works in those definitions (e.g. semantics of encapsulation).*

- We have expanded on the description and intuition of the $\triangleleft$-operator, provided a better description, as well an example and a figure demonstrating how it is used for variable rewrites (section 3.3.2)

3: *Streamline Section 2.4.*

- We have significantly restructured and rewritten Section 2.4 (now Sections 2.5 and 2.6). We initially followed advice from the reviewers to take a more top down approach to the the description of the system, but we found that this did not suit the complexity of the system and ultimately obfuscated detail rather than clarify. Instead we have attempted to capture the intent of the reviewers' advice, and present the system bottom up, however starting from a more abstract description that includes the three main insights, and a figure demonstrating the structure Necessity. We believe that Section 2.5 now presents a much clearer picture of our system. In Section 2.6 we apply the parts described in Section 2.5 to develop a proof.

4: *Avoid sudden/unmotivated topic changes.*

- The reviewers correctly identified that certain aspects of the original submission lacked flow and changed topic frequently. This mainly occurred in the introduction (with sudden switches to related work), and in the description of our case studies in Section 3. We have reworked the introduction with the reviewers' advice in mind, and have essentially organized it into three parts: The problem we are tackling, relation to earlier work, our contribution. We introduced related work in a way that does not interfere with the flow of the paper.

5: *Comment on calls from internal to external modules.*
- We have included an explanation that most of the related work does not support external calls (lines 260-263), have discussed in more depth in related work (lines 1198 onwards) and outlined our thoughts on possible solutions in lines 1201-1203.

6: *The Coq proof should be completed: currently it has so many admits.*
- The reviewers requested that we make the following changes to the Coq proofs:
  - **P1:** In the original Coq proof there was an admitted proof in a file that was not used by the main Coq model. We proposed to address this by deleting the file.
    - we have deleted the file
  - **P2:** The original Coq proof included some admitted proofs of properties of an assumed specification language for the purposes of proving examples. We agreed to rename these proofs to `Hypothesis` per the reviewers' request.
    - we have renamed the the proofs
  - **P3:** The original Coq formalism included a proof involving some variable renaming that we had admitted as it was not central to the main results. The reviewers advised us to complete the proof.
    - We have completed the omitted proof
  - **P4:** The reviewers advised us to include a description of each Coq file, and why different properties have been admitted
    - We have included this description in the readme.

7: *The motivating password example is still not realistic. The example does not provide any interface to allow someone to get the correct password, or at least initialize the password. I think the example should be extended in such a way, which may require a more powerful logic. If this doesn't work, the paper should provide another example and clarify the weakness of the logic.*
- As we already discussed with the reviewers after the rebuttal period, the language Loo does support object initialization. To demonstrate this, we introduced a method init in lines 136-137.

2. Strongly Recommended Changes

8: *Clarify what you mean by "emergent behavior".*

- We have included a description of *emergent behaviour* in the new introduction, lines 40-43.

9: *More clearly separate introduction from related work. In general streamline flow of the paper.*

- We have followed the reviewers' advice, and this has greatly improved the flow and conciseness of the introduction.

3. Recommended Changes

The reviewers also requested that we address all of the other minor changes requested by each reviewer. Here we detail those changes.

3.1. **Reviewer A.**

(1) *In l. 36-55, the introduction suddenly becomes/mixes with a part on related work. To my opinion, this completely distracts from the description of the problem that this paper is attacking. I'd rather that this part is moved to the dedicated related work section.*
- We have restricted the related work in the introduction to only lines 13-15 and 55-65.

(2) *l. 68. At this point (and also many points later), it was completely unclear to me what "emergent behaviour" is supposed to mean.*
- We describe this in line 41

(3) *Section 2.4, I must say, I find extremely tedious and very difficult to follow. I firmly believe that this can be streamlined to that one must not mentally follow through 9 steps (a - i). Many notions are also only explained later. E.g. it is not clear (and also does not become apparent from the explanations in Sec. 2.4) why one needs to construct from per-method conditions the single-step conditions. And is this really important in order to get an overview of the approach that the paper is taking?*
- We have separated the notion of assertion encapsulation, and then described the three parts of the proof system.

(4) *.102: I have two problems with the notion of being encapsulated: - Maybe a minor comment, actually, but again - I'm sorry - I do not understand the name "encapsulated". The fact that the concept "only by executing a pice of code C one can invalidate a logical assertion A" is called "C encapsulates A" does not make much sense to me because "encapsulating" - to me - suggests rather that A is somehow part of C or that A is somehow wrapped/surrounded by C.*
- More severely, I still cannot get my head around the notion that only by executing C one can invalidate A. Imagine `C'=C;x:=x`. Obviously `C'`$\neq$`C`, but if `C` can invalidate `A`, so can `C'`. How can there be a piece of code `C`, so that only `C` but not `C;x:=x` can invalidate `A`?
- The invalidation is done by methods of the module, not by a single, specific piece of code.

(5) *says "Note that our proofs of necessity do not inspect method bodies" This makes absolutely no sense to me. How can I infer - or more: prove - anything about an object C (the code) without looking at C? This needs an explanation. In the explanation that follows you mention pre and postconditions of methods, but how can I prove pre- and postconditions of methods, if I cannot look at the methods?*
- We assume that the proofs of the pre- and post-conditions come from some external system. We have added a Fig.1 in which the functional specifications are indicated in gray, indicating that *Necessity* is parametric with these parts.

(6) *l.263 following: At this point I was wondering, which of Mod1, Mod2, Mod3 is internal, what is external? It would be good to refer back to that example and point out to the reader what is supposed to be internal and what external.*
- We now give descriptive names to the modules.

(7) *Def. 3.2: Why would one write Arising(M, Y, sigma) iff ... Y; M, sigma*$_0$ *... Why flip the order of M and Y? Does that not cause unnecessary confusion? Or is there a good reason to flip the order?*
- There was no good reason. We have adopted the order you suggested – thank you.

(8) *Def. 3.8: It would be good to ostentatiously clarify that necessity specifications cannot nest, i.e. the nonterminal S does not appear on the right-hand side of the grammar. Only nonterminals A and those come from the language Assert, I suppose.*
- This, indeed, is crucial. It follows from the syntax, since S (necessity specifications) do not nest in Def. 3.8. We stress this in lines 86-89.

(9) *l.442-456: It is totally unclear how/why the* $\lhd$*-operator does the trick for you necessity modalities. Part of the reason is that the definition* $\lhd$ *is described/explained in not enough detail (for me). But I also firmly believe that the definition of the semantics of the necessity modalities deserves to be provided some intuition. In particular an intuition of how* $\lhd$ *defines these semantics.*
- You are right. We have taken your suggestions at hand, and gave a longer explanation, and a diagram, in §3.3.2. We were also able to simplify the definition. Thank you for pointing this out.

(10) *l.482: I believe that it deserves an explenation why no module satisfies* $\texttt{NecessityBankSpec}_c$*. It is not obvious to me.*
- We have expanded section 3.4.1, and have given explanations for the cases where specifications are not satisfied.

3.2. Reviewer B.

(1) *90: this is the first example I noticed of a weird transition. The authors go straight from saying that there are three new operators to a lot of detail on the first of those. The paper here would flow better if the authors gave an informal description of all three operators (to give the reader an intuition for what's coming next) rather than going straight into gory detail.*

- we took this suggestion at heart, and gave an informal introduction to the operators in section 2.2, and then used them in section 2.3

(2) *99: the authors state that necessity operators are second-class, but don't really justify this choice or explore its consequences, and it is never returned to. I'm not sure if the necessity operators being second-class is actually the right choice, and especially at this point in the paper, where I as a reader don't yet fully understand them, this statement throws me off.*
  - We now explain that this is what allowed us to develop the proof logic – lines 86-89.

(3) *200: when you make an assumption like this one, please justify it to the reader rather than simply saying "Note ..." Especially for such an important assumption, as this one, it is unsatisfying as a reader to be left wondering why you have done this.*
  - We now explain that this is in agreement with lots of related work

(4) *205: you refer to Mod1, Mod2, Mod3, etc. many times, but their names are not descriptive at all. This was the point where I got mildly annoyed at having to go back and check which implementation Mod3 was. I suggest renaming the modules to something that describes their properties, e.g. Mod3 could become "SafePwdSet" or something like that.*
  - We have now given them more descriptive names, i.e. $\texttt{Mod}_{good}$, $\texttt{Mod}_{bad}$, and $\texttt{Mod}_{better}$.

(5) *243: you might consider emphasizing this point more: it is a strength of your approach that it doesn't make many assumptions about how these annotations are checked, and so is compatible with lots of existing work.*
  - Thank you. We added such an argument in lines 99-102.

(6) *351: I have a serious concern about definition 8 (x access y) here. It seems to me that it is saying that if y's value is exactly the value of one of x's fields (f), then x can access y. That seems reasonable, but I don't see any notion of multi-step access paths here, which seems suspicious to me. After all, intuitively I would expect that (x access y) would be true if for example the value of the expression x.f.g is the same as the value of y (and so on for any arbitrary number of field accesses). Did I miss something about your setup/language that forbids fields themselves having fields? Or is it the case that (x access y) is defined to be false in a case like the one I mentioned above? If so, that seems to me like a "soundness" problem in the sense that it violates what I as a specification writer would expect that specification to mean, and therefore might result in incorrectly-specified code.*
  - We now clarify that access is not deep, and why it should not be. C.f. lines 484-487.

  382: inside is a very useful concept. I might consider mentioning it earlier, perhaps by working it into one of the examples in section 2.
  - We experimented with that, but we found that this would require too many explanations too early, so we gave up.

(7) *459: this is another case where the flow of the text is jarring. There is no explanation for why now is the appropriate time to consider more examples of specifications. This might make more sense as a separate section rather than as a subsection, with a short justification saying something like "we now present some examples to give the reader an idea of the expressiveness of our approach"; you didn't explain that that was the goal of the example section in this draft until the very end of the section!*
   - Done

(8) *1036-1042: there seems to be a lot of overlap between the present work and Chainmail. You might want to spend another sentence or two here discussing the differences.*
   - Differences are now described in some depth in lines 1163-1174.

3.3. **Reviewer C.**

(1) *Section 1: I'm not sure if paraphrasing liveness and safety is a good idea because liveness/safety verification in the traditional sense is also reasoning about sufficient conditions for good things to eventually happen or for bad things to never happen. The point here is the distinction between sufficient and necessary conditions about the behavior of a program.*
   - We have drastically reworked that part, and no longer talk of liveness.

(2) *Section 2.4 doesn't work very well (at least for a first reading) because it is written in a bottom-up manner. I had no idea why assertion encapsulation is the first step because I didn't have a big picture how necessity specification might be verified. Explaining backwards from Part 4 to 1 may work better (but I'm not sure...).*
   - We have tried this, but it did noit work that well. Nevertheless, we have re-worked Section 2.4 (now 2.5) and hope it is mush better now

(3) *It seem to me that it can be hard to show assertion encapsulation because of universal quantification over all external modules and states. The existence of a type system and a proof system for assertion encapusulation, which is discussed in 4.1.3, is nice and plausible but I find the discussion on Enc(A) handwaving and the derivation in Section 5.1 is hard to follow. I think you should explicitly state that the bank account example is typed by the assumed simple type system for confinement.*
   - We say that now in lines 318-319

(4) *I'm not sure why Section 5 extends the simple bank account example so much. Is the extension needed to demonstrate the verification framework, or you thought the examples in Section 2 would be too simple? Maybe they are simple but I think it is a good exercise to show verification of the first examples. It was hard for me to follow the extended example as a first verification exercise.*
   - We now prove the original example in Section 5, and the extended example in the appendix. The reason we also prove the extended example is in order to show the use of ghost fields, and encapsulated classes.

3.4. **Reviewer D.**

(1) *First, the way to connect the "from" and "to" conditions seems to involve unnecessary complications, which may restrict the applicability of the work to other languages. Updating physical states using fresh variables sounds too restrictive. For example, in assembly, one cannot create fresh registers. Here it is unclear to me why the authors cannot use logical auxiliary variables as done in modern separation logics.*
   - We are familiar with the technology proposed, and we could indeed use it for the semantics of "from"-"to"-"onlyIf" and for the semantics of "from"-"next"-"onlyIf". But we could not find how to use it for the semantics of "from"-"to"-"onlyThrough". Moreover, the Coq formalization uses the adaptation operator, and so the change proposed would not be easy yo accommodate. On the other hand, we have vastly improved the explanations of the adaptation operator.

(2) *Second, the allowed control flow between the external and internal modules is restricted (ie, the internal module cannot invoke external modules). Since reasoning about the internal module should be essentially independent of external modules (because external modules are completely arbitrary), similar reasoning as presented in this paper should be possible even when the internal module invokes functions of external modules.*
   - We discuss more the issues around calls from internal to external objects in lines 260-263, line 1198 onwards, and outlined our thoughts on possible solutions in lines 1201-1203.

(3) *Finally, more importantly, although the paper claims the (S3) condition instead of (S2) as an advance over Chainmail, the condition (S3) is not so convincing. There are two concerns. First, (S3) does not say anything useful in the presence of an external module that has access to the password, which is the case in general. In other words, (S3) cannot distinguish between external modules that have access to the password and those that do not. On the other hand, (S2) implies that external modules without knowing the correct password cannot change the balance. Second, (S3) seems to relying on the setting that the password is an unforgeable object instead of a string, which is rather artificial. On the other hand, (S2) seems to work even when the password is a string.*
   - We discuss this in section 2.3.1.