Improvements to OOPSLA-128

A. Overall improvements as promised

We use > to indicate what we had promised,
We use *** to indicate what we have done

> Expand Section 2 to better explain scoped invariants (RA), and illustrate the Hoare logic in proving external calls (RC).

> We will ensure that all terms are explained before use -- either informally, with a definition, or through application to an example -- including terms such as path, frame, access, external state, current call, step, object reachable, state reachable, and pointer semantics.

** We have enhanced section 2, to better illustrate the problem as well as our approach. We included a heap diagram on page 3, illustrating the problem. We added state diagrams in Fig. 1, illustrating “current point of execution”, and added further diagrams in Fig. 2, illustrating “protection”. Moreover, for ease of understanding, throughout the paper, all illustrative examples are based on the example on page 3, or Fig. 3 -- except for the very specific matters on lines

** We have included definitions of the terms direct access, indirect access, frame

** We have all read through the paper checking all key terms are defined before use

> Provide more examples and motivation e.g., discuss how reasoning about access allows us to reason about internal fields;

** We have added explanations, motivation, and example as follows

** Motivations
- 165 - 188: motivation for “protected” and for scoped invariants
- 240-247: motivation for “protected” and for scoped invariants

- 265-284 reasoning about access allows us to reason about internal fields
- 409-416 reasoning about access allows us to reason about internal fields

- 646-560 design alternative for protection

- 761 - 776: further motivate the design of scoped invariants, and compare with History invariants and object invariants

> take further examples from the appendix to explain the applicability of our approach.

** Created section 7.1 to summarise the applicability.

B. Summary of Changes Section-by-Section

1. Introduction

- Streamlined and simplified.
- Added an introduction to object capability model (50-61)

2. Approach

- explained the design space for our specifications 187 - 238, to motivate the novel concept of “from the perspective of current execution”
- Added illustrations - eg a heap diagram on page 4, current execution, Fig. 1, illustrations of protected Fig. 2, illustrations of scoped execution Fig. 3
- Added explanations as to how underlying Hoare Logic is expanded to deal with scoped invariants and protection -- section 2.2
- Added explanations as to how the external call inference rule can be applied to prove S_4 -- section 2.3
- The illustrations in Fig. 2-4 came from the Shop example, Moved more illustrations (of more intricate situations) to appendix

3. Underlying Programming Language

- Added syntax of the language and operational semantics Fig 4, and Fig 5
- Removed the Fig about execution as it already is part of the Section 2
- Explained the applicability of our approach -- lines 421-422

4 Fundamental Concepts

- Moved two lemmas from the end of section 4.1 to the appendix. These lemmas are not necessary for understanding what comes, nor are they used in the proofs later on
- Removed the Figure about reachable objects to the appendix. This is not a difficult concept, and given that there is more material in section 2, the reader will have understood by now.
- Added the concept of “reachable-from”. This allows us to simplify Def. 5.4 later on
- Added Def. of Well-formed state, which was implicit in the past, and crucial for lemma 4.6, and 8.3

5. Assertions

- Simplified the Def. of protected.
- Removed the Fig. about protection, and rely on Fig. 3 instead.
- Added discussion of alternative Definition for protection

7. Specifications

- Explained difference to Object and History Invariants in terms of a diagram
- removed the diagram, and refer to Fig. 3 from Sect 2
- Added a discussion explaining why we need both a postcondition as well as an invariant part

- Added section on Expressiveness

8. Hoare Logic

- Moved the discussion of adaptation to here
- Simplified example demonstrating the need for the external requirement in Lemma 8.3.3 and 8.3.4
- Added Discussion on Polymorphic calls -- and more details in Appendix H.6
- Added Example 8.5 demonstrating Hoare logic proof of the external call
- Added Example 8.6 demonstrating how to prove that “set” from M_{fine} satisfies S_2, and more in appendix H.5, and why we cannot prove that “set” from M_{fine} satisfies S_2.
- Added section 8.4, which outlines the Coq proof of the example

9. Soundness

- Summarized this section to only touch upon the subtleties of this proof

C. Detailed Response to Reviewer_A

We use >> to indicate what the reviewer had written,
We use > to indicate what we had responded,
We use *** to indicate what we have done

>> was expecting a paper on FFI rather than intralanguage safety;
> the only requirement is that a module’s private fields are not visible outside the internal module …

*** done, cf 421 - 424

>> .. to introduce the execution diagrams from later in the paper a bit earlier, … to make the validity of a scoped invariant clearer with the visualization.

** great suggestion, done in 259-267

>> … but Mgood is a simple use case; practical plugin APIs can be considerably more complex with many more moving parts and abstractions.

>> Further, Appendix E contains several case studies chosen from the literature.
We would be glad to consider patterns that you could suggest.

*** Mgood is a minimal, well-chosen example, demonstrating common patterns in capability-based security. Our work is unique in that it offers a Hoare logic for external calls. More in Sect 7.1

>> It would be good to talk in the related work about how this system could be used to further prove correct other abstractions or program verification tools that could be more automated, for example, or how it could potentially be checked automatically.

** The current paper is devoted to the logic; whilst we do have ideas about toolsintegration with tools, and have discussed some of them in the response from December, we decided not to include in the paper, as these are, indeed, further work.

>> A very minor comment on setting expectations; from the title, I was expecting a paper on FFI rather than intralanguage safety; as I think that it might be possible to make the validity of a scoped invariant clearer with the visualization.

> The paper is also about interlanguage safety (c.f. RB2's comments), provided that the platform offers means to protect a module's private state; cf capability-safe hardware as in Cheri.

** added to lines 421 - 424

>> In your view, how can we scale verification of defensive consistency? …

Our response to the review contains our thoughts on that matter. However, since tre paper is about a new logic, we do not discuss integration with tools in the current paper.

D. Detailed Response to Reviewer_B

>> Section 3.1 defines the language as "a small, imperative, sequential, class based, typed, object-oriented language." Therefore, I assume such a language supports all standard components of popular OO PLs, such as inheritance, dynamic dispatch of method calls, exceptions, and so on.

** In line with similar research, eg [32,53,94]we develop a minimal calculus that allows us to demonstrate our ideas. We clarify this in lines 420 and 424-427.

>> A state is defined as a heap and a stack. At first, I misunderstood the stack, since I assumed it was the heap+stack memory like C++....

** Talking of stacks of frames, line 199, and showing them diagrammatically in Fig.1

>> where each frame contains the values of local variables of one method. Then, each frame "consists of a local variable map", but it stays obscure to me if such a map relates local variables to primitive values and addresses …

** Frames formally defined on line 401, and note that values are addresses

>> indeed or if it might map local variables to objects (aka, mapping from fields to values) as

well (in this way, something like stack-based allocations of objects in C++ might be supported)

** no stack-based allocations, see Fig. 4. Note that stack-based allocations can be encoded, though

>> quite surprised about the definition of private fields and methods that is provided: it looks like private components are available to all classes inside the same module. This is definitely not the standard semantics of private components, which are only accessed by the same class or object but not by other classes belonging to the same module.

** Indeed, there are many possible definitions for private fields, and we have chosen the weakest. But for the purposes of our work, all that matters is that external objects cannot read/write private fields, and thus we chose that definition. All the results of the current paper would still be valid if we had chosen per-class privacy. This is explained in footnote 9.

>> The top stack contains the next statement to be executed, which is fine until exceptions are thrown, but as soon as an exceptional state is raised, such an approach seems not to be in a position to deal with that.

** We do not support exceptions, as stated on line 428. However, supporting exceptions would not be problematic for us. We would only need to pop the frames from the stack until we find the corresponding exception handler. Cf Ancona et al, “A core calculus for Java exceptions”, OOPSLA 2001.

>> Statements are assignments to variables and fields, allocation of new objects, method calls, and field reads. However, what is the behavior of a statement like "new C();"? Just allocates an object that is unreachable after the statement?

** Adding syntax and operational semantics should deal within that question. In particular, “x := new C();` is a statement. Indeed, it creates a new address containing a new object of class C, initialises its fields, and assigns the new address to x. This is a standard treatment, and we did not think it necessary to outline in the main paper. Happy to do that if the reviewers want us to.

>> … as far as I can see, the state does not capture anything about the dynamic type of variables, how can you know the continuation of the body of the method?

** Objects are pairs of class, and field maps, cf line 402

>> Later on (def. 5.3), an oracle classOf is used, and this oracle, starting from a state and an address, returns the object's dynamic type allocated at the given address. Unfortunately, such an oracle is neither formalized nor discussed in the paper.

** Because such a classOf function is standard, we did not elaborate in the main paper, but do describe in the main paper, and refer to appendix

>> …(e.g., no idea about exceptional states in def. 4.1).

** external state defined on line 101

>> Unfortunately, the absence of a (even minimal) language and semantics opens the door to assumptions and ambiguities that compromise the understanding of the paper's formal content.

** Moved syntax and operational semantics of the language to main paper, even though they are rather standard

>> Another major problem/doubt is how the proposed approach resonates statically on internal and external calls. In particular, in a (polymorphic) object-oriented program, we might have method calls that, in some executions, call external methods and, in others, call internal methods. This is not a particular or exceptional situation; it happens in most real-world OO programs. Consider, for instance, just logging where a logger writes the messages somewhere. Such a component is intrinsically polymorphic since, during the development and testing phases, one wants to log everything locally, while during deployment, only a few messages are often logged through external services. Therefore, the logger is initialized with objects instances of different classes, some internal and others external.

** Excellent point. While we do not support inheritance, we do support dynamic dispatch and thus support the scenario above.

>> When I looked at Fig. 7, I realized that such a scenario is not supported ….

** We can support this scenario, even though the rules in Fig. 7 do not directly address the possibility that the receiver might be internal or external. This need not be done through rules in Fig. 7, because we have case split in the Hoare logic. We discuss this in lines 944 -947, and H.6.

E. Detailed Response to Reviewer_C

>> 1) Not a self-contained paper:

>> (i) The formal material critically depends on the appendix. …

** We moved syntax and operations semantics of Lul to action 3 in the main paper.

>> (ii) The proof of why Mgood satisfies S2 is also in the appendix, …

** We added to Section 8 two examples, Example 8.5 show how external calls can be proven, and Example 8.6. outlines how set from module M _fine can be proven, and why set from module M _bad cannot be proven. Appendix H now contains proofs for all methods.

> Worse, the appendix does not cover neither S3 (and in
particular the S4-related step for the external call), which is the actual
property we care about (see also point 2 below),

** All this is now available in appendix H

>> nor discusses why S2 does not hold for Mbad, which is important as a demonstration of how the proof system handles problematic situations.

** This is now in Example 8.6

> > I cannot figure out how the rules of the logic can be used to show that an invocation of an external method implies a post-condition that specifies something about the value of an internal field.

** This is now in Example 8.5

>> In general, due to the absence of specifications for external objects, I cannot see how the proposed approach can help reasoning about high-level properties besides access to objects.
** We hope that the discussion of the proof of (1) on page 8, and Example 8.5 address this

3) Evaluation of the approach:

>> ..the evaluation of the practical applicability and limitations of the idea
is almost non-existent. …. Another, substantial
case study would help a lot with that front. If selected carefully, it could
also help with demonstrating the delta with some of the other
approaches mentioned in the paper.

** The running Shop example in this draft is carefully chosen to be minimal, yet intuitively comprehensible, while still embodying the key difficulties within the problem. More examples in appendix E.2 -- these are the kind of examples the current state of art can tackle.

4) Presentation:

(i) Section 2 references terms such as path, frames and steps that have
a specific meaning in section 3 but do are not explained in section 2.

* We endeavoured to explain all terms before using, thank you for pointing out

>> (ii) The formal sections are pretty terse. Most of the space is occupied
by definitions and the examples presented are not really explained….

** We have examples in most sections, streamlined the examples to refer to Shop in most cases, and supported them by diagrams

>> … have not found an example that shows concretely why the quadruple for specification needs an extra dedicated assertion-guarantee for external method invocations.

** This is an excellent question; thank you. We have added such a discussion in the paper, supported by an example. Lines 778 -790

>> In an attempt to be constructive, I think it would be a significant
improvement in terms of presentation to:
(i) Turn section 2 into a ``the logic by example'' section. Similar to
the current version, the section should focus on the Shop example, …

** This is an excellent idea, which we have adopted. We have made the section 2 much more accessible, and used diagrams from the Shop example to explain “protection”.

>> (ii) Remove all material from the main body that have to do with the
soundness of the logic, and use the recovered space to give a complete
self-contained account of the remaining formalisms.

** We have considerably shortened the discussion of soundness

>> (iii) Bring in the paper some of the discussion about expressiveness
that is now in the appendix.

** Done! Section 7.1