——————— REVIEW 1 ———————

R1: The paper is generally well written and easy to follow. There are some aspects that need clarification: especially Section 4 needs work (perhaps by addressing omissions in the Sections introducing it). The contribution is very unsurprising, but I'm seeing that as a good sign. There is an interesting error in the last of the examples which was not caught by verification and could not be caught by any specification. I do not believe the authors put it in intentionally and it raises the question of how useful this approach would be in practice.

A: See below

More detailed comments: the work is closely tied to reference [13], which is by two of the same authors. Still, the difference between what is presented in this work and what is considered previous work needs to be made clear. The last three lines of the first paragraph of the introduction leave some confusion.

A: Fixed the last three sentences. We think the contribution is clear.

In the second section, Kleene Algebra is introduced, and the authors write that $\alpha\beta$ is often written instead of $\alpha \cdot \beta$. In Section 4 and onwards, $\alpha\beta$ seems to refer to function application, which looks so similar that I suggest not to leave out the dot.

A: Fixed.

After proposition 2.1, the paragraph needs some clarification: "We use an alternative approach", alternative to what? Especially: is this alternative approach given in [2] or is it an alternative to [2] ? Also for "these allow only one type parameter", perhaps repeat 'type classes' instead of 'these'.

A: Fixed.

More crucially: why is that a problem? What do you need more type parameters for?

A: It is not a problem but a design decision.

Another question that came up, is whether $n = tn$ or $n = n^3$ is the defining requirement here for antitests.

A: No.

I don't really see the antitests in the remainder of the paper either.

A: Some readers may wish to check the Isabelle files. They might find this comment useful.

While reading Section 3, I would have liked to see composition (or ;) introduced right at the beginning, to help me verify that I am understanding the intention behind the KAT operations.

A: Fixed.

When introducing 'inv', add 'we define'.

A: This definition is already indicated by putting "invariant" into italics.

For (h-inv-plus), consider changing the left hand side $\wedge$ into a $\vee$.

A: This is unsound.

At this point, it is unclear whether we are working in an abstract KAT, or one in which $\cdot$ is Kleisli composition. By the time we get to Section 4, we seem to have moved to the latter, but this is not clear on the first read.

A: It is clearly indicated before the formula block that $\alpha \in K$ and $i \in B$.

In the third to last sentence of Section 3, a reference to h-loop-inv is made, which I did not find in Section 4 although the phrasing suggests an explanation about it (especially why programs need it) should be there.

A: (see Section 4) refers to hybrid programs, not to the inference rule.

In the introduction of the evolution commands (first line of Section 4), the type of f is unclear.

A: Type added.

Section 3 uses the symbol f for programs, or elements of C.

A: Discuss with Simon.

The embedding of tests ?P does not seem to be used (or further explained) in the paper.

A: Clarified this.

On a second reading of the paper, it is clear that the evolution commands are explained by the paragraph starting with 'Systems of equations', but that paragraph does not even mention the term 'evolution commands' ands takes the reader for a tour through definitions of trajectories, orbits and IVPs, before showing (st-evl). The explanation of assignments seems to be split between the second and third paragraph, and the paragraph following (st-evl).

A: Second and third paragraph explain how lenses model state spaces. This is then used to explain evolution commands and after that assignments.

You write 'Lenses x and y can also be checked for independence'. Here, the notion of independence needs to be explained (or at least make explicit that you are not referring to linear independence).

A: Added some words.

In the last sentence you write that Euclidean spaces 'can be' supported. Are they supported? If not, why not, what was the difficulty? (Perhaps this does not need to be mentioned, currently it raises questions).

A: Clarified this.

You write that T is an open interval. This puzzled me a lot: many of your examples seem to start at 0 (including 0), making the intervals non-open (they might still be open at the other end). Especially you claim that downward-arrow-t is usually the closed interval [0,t] seems to contradict that T is open. Also, I don't understand why you would need T to be open.

A: Open intervals may contain closed intervals as subsets. T is open as a requirement of the Picard-Lindeloef theorem.

In introducing the terms 'trajectory' and 'integral curve', use the word 'the' to disambiguate which of these terms apply to whether the term applies if it is uniquely defined.

A: Fixed.

On the next page, you introduce the dagger symbol for state updates. I don't believe this is used elsewhere in the paper. In addition to a reference to [7] where one might find 'the expected laws', perhaps a reference to an Isabelle theorem name might help the reader to find out which laws you are referring to exactly (without needing to repeat them in the paper).

A: Done.

You write: 'Such laws can be applied recursively for symbolic evaluation of deterministic programs'. The addition of the word deterministic suggests that this cannot be applied to programs that are non-deterministic, yet your examples show non-deterministic programs. Also, the laws seem to apply to non-deterministic programs as well. If you intentionally added this word, you will need to explain your intention to the reader.

A: We should have written 'hybrid'.

The first paragraph of Section 5 ends with a sentence in which a lot of the concepts of Section 4 come together. At this point, it might be nice to have a sentence that starts with 'In other words, ...' so that I can verify that I understood everything (my guess at this point is that the evolution command executes for some non-deterministic amount of time, such that q must hold for some time after 0, and the guard should force the program to stop).

A: We have revised this extensively across the paper, mainly in Section 4.

In the definition of guarded orbitals, your guarded orbital definition seems to rule out that I would create guards for evolutions in which at least one second passes, though of course I could still write a while loop in which the condition ensures the passing of at least one second. However, I am allowed to choose a U such that at least one second passes for every evolution?

A: T is given by the Picard-Lindeloef theorem as the maximal interval of existence of a solution, we are locally Lipschitz. The set U allows us to pick an subinterval of T that we care about. It adds expressivity to the approach. Guards do not generally talk about time but we can easily add time as a parameter in our differential equation.

Why does your logic use the downward-arrow-t (which is even based on T rather than U), it seems like this intentionally complicated your semantics, but that intention is not clear from the paper.

A: Fixed.

In the specialised guarded orbits, the last t (fourth character from the right) should probably be a tau.

A: Fixed.

In (st-evl-flow), the right hand side contains the free variables U and T, and the left hand side contains the free variable x (or perhaps x itself is syntax here?).

A: Added a sentence.

I need some help in understanding when these variables are bound: for instance, is U an essential part of the operational semantics? Does it determine a range for the system by which it gives execution back to the program?

A: We do not have an operational semantics and U was explained.

In Example 5.2, you write that c must be in the set $\{0, T_u\}$. This is a closed set, not open as required right above the example, which leads me to conclude that 'f a c' itself must be of the type $S \to S$ (indeed, abbreviation dyn supports this reading). However, not knowing how to interpret [ ] with $\mapsto_s$, I fail to understand how to match the symbols in the example to the h-evl definition. Also, what is the intention behind T_u?

A: This is a 2 element set, not an interval.

Then in the example, $on\{0..\tau\}UNIV@0$ shows up, it seems that some of the free variables I was concerned about earlier are actually determined by the syntax of programs, yet left out in the syntax descriptions so far?

A: We preferred to simplify the mathematical presentation previously. Most of the 'free variables' used before were implicitly quantified through the text. There is definitely a mismatch between traditional mathematical specifications and the functional programming style imposed by Isabelle.

What does the bold U mean? Does it bind T?

A: Explained in the next paragraph.

In Lemma 6.2, I feel like I would need to know more about how ¡ is defined on differentiable functions in order to verify part 3 and 4. The inclusion of these properties also makes me wonder about what is intended in part 2. In typical interpretations, $u = \lambda x.x$ and $v = \lambda x. - x$ should be counterexamples to 3 and 4 for S = any open interval on R that includes 0.

A: Need to prove in Isabelle.

In abbreviation dyn in example 6.3, DINV is used. This has not been defined operationally. I see no reason why it would differ from INV (or why it would need a separate symbol).

A: Fixed.

In lemma tank-diff-inv, what is Guard? (Is it a free variable, syntax or a constant?)

A: Done.

In Example 7.3, you use ?R instead of .... If this is for clarity, perhaps use three different variables, ?R1, ?R2, ?R3.

A: Fixed.

To the last sentence of Section 7, add 'Correctness of' (the program itself is not obtained, it was always tank-dinv).

A: In fact, the final program is constructed in this proof by refinement, and it is correct by construction. We further emphasise this in the text.

In Section 8, abbreviation bb-evol, the condition is 'v = 0', although the text suggests that this should be 'x = 0'. This error in your program will allow the proofs to go through, so it is a good example on how you may still have errors in your programs even though the programs satisfy their specifications. In particular, your use of guards will typically cause many of the desired correctness properties to go through for faulty programs. This, in my view, is a fundamental weakness in using evolution commands to model hybrid systems. As a counter-argument, it might be merely related to not having termination proofs for these systems. Feel free to speculate on this in your discussion.:

A: We have fixed this. The referee should not be worried, we simply verified another program that makes physical sense, but not a specification that should not be correct.

——————— REVIEW 2 ———————

paper presents hoare-like logics for the derivation of hybrid programs/systems, inspired from morgan's calculi of program construction. models interesting aspects of programs and specifications, and calls upon interesting mathematics to generate and prove verification conditions. I did not check all the mathematical details (some of which refer to research contexts I am not familiar with) but the parts I could follow look credible and interesting. I have two questions that may require clarification:

- what is a hybrid program? the programs that the paper studies are very different from the kind that morgan investigates in his book, but I could not tell which of these differences makes them hybrid. neither this paper nor some of its bibliographic references has an explicit definition.

- what do the statements "while t inv i do alpha" and "loop alpha inv i" mean? how does their semantic definition differ from "while t do alpha" and "loop alpha"? what happens if i is not actually an invariant of the iteration? does the statement become abort?

A: Nothing to do.

——————— REVIEW 3 ———————

Differential dynamic logic (dL for short) is a logic used to specify and verify properties of hybrid programs (programs combining discrete and continuous behaviours).

This paper introduces a differential Hoare logic (hL) and a refinement calculus (rL) which can be seen as subsets of dL. The motivation of the authors is that these logics, certainly less expressive than dL, are 1) usually enough to express specifications that are needed in concrete applications 2) are easier to implement.

In verification of dynamic systems, the deductive method is the most commonly used. In this perspective, the authors provide deductive rules of hL and rL and an implementation in Isabelle.

In principle I find attractive the idea of developing a Hoare logic for Hybrid systems. However, the paper suffers from many problems

-The contributions are not clearly stated and many definitions are missing.

A: We think the contributions are stated clearly. We would have been happy to provide further definition, had the referee provided some examples.

-The deduction rules of hL and rL are dispatched along the paper. The proof of their correctness is not clear to me. I guess that this is the content of lemmas 5.1 and 7.1 for example, but again this is not clear from the statements.

A: We added some text to clarify this.

-The authors say that they found hL and rL easier to implement in Isabelle. This is certainly true, but this is not a proof that those logics make verification of hybrid programs any simpler. Does the verification process become faster? Benchmarks would have been useful to support this work.

A: We have not made any claims as to the efficiency of the verification process apart from automatic verification condition generation. As usual with program verification, this is the easy bit. The verification of data domain verification conditions can be highly non-trivial and very interactive, as our examples show. We are not sure that benchmarks make sense in this setting.

- Are the deduction rules complete?

A: We do not know and this has not been claimed in the paper.

In sum, the subject is interesting, but the paper would benefit from a deep reorganization and rewriting.

A: We would have very much appreciated any hints on how this could have been achieved.

——————— REVIEW 4 ———————

This paper presents Hoare Logic and Refinement Calculi for Hybrid Systems modeled as Hybrid Programs. The work is very much inspired by Platzer's differential dynamic logic. All the theorems and proofs are formalized in the Isabelle theorem prover.

The paper represents an important contribution, fully formalized in Isabelle. It is well-written and interesting.

However, I am very confused by Lemma 5.1. In particular, it looks like if I take U to take the empty set (which I am allowed to do, at least in my understanding), the precondition is vacuously true and I can always prove the postcondition. Something must be missing here, please fix it. I am quite concerned about this technical issue, and I don't understand how you can possibly prove Lemma 5.1 state like this.

A: Thank you very much! We forgot to add the condition $0 \in U$ to Lemma 5.1. It is present in our Isabelle formalisation. We have now added it and the lemma should be fine.

Could we do the same things without lenses? They don't seem essential to your development, i.e., the power of lenses seems overkill for what you're trying to do.

A: We could do the same things without lenses, but they simplify the integration of various store models which is highlighted in the introduction. As we aim at an open modular framework (see conclusion), we deem them very valuable.

Section 4, where is the full state transformer semantics for all operators of the hybrid program? It would be really interesting to see if fully fleshed out.

A: It is in Section 3 and Section 4. We have added text to clarify this.

Lemma 5.1 (and the subsequent lemmas) can only be proved sound with respect to a semantics, but you never explicitly provide a semantics.

A: We explicitly provide a state transformer semantics, but have perhaps not clarified this well enough around Lemma 5.1. We have now added an explanation.

Why are you developing a Hoare Logic when a Dynamic Logic already exists? The Hoare Logic is less powerful, but also has less complexity (but you never make this argument explicitly in the paper).

A: See introduction and conclusion.

Still on Lemma 5.1, why do you take a T and a U and how are they defined in practice?

A: We have added an explanation.

In my understanding, dL always imposed T and U to be the same, and it imposes it to be [0,+oo) or whatever is the longest interval where the flow is defined.

A: Traditional dL considers dynamical systems where solutions are assumed to exist for time intervals given by the positive reals. We assume a more general setting of locally Lipschitz or even continuous vector fields, where solutions have weaker properties.

Why do you restrict yourself to cases where the Picard-Lindelof's theorem is applicable? dL doesn't do that explicitly. What does it buy you here? It's unclear where you use this assumption.

A: It should be clear from our paper that we don't: we use invariants in situations were even unique solutions need not exist. Traditional dL is even more restrictive in that not even solutions involving transcendental functions can be used explicitly.

In Lemma 5.1, please write down explicitly that P:S-¿B, Q:S-¿B and G:S-¿B. I got confused for a while (at least P is not mentioned). Please also explain the $(\lambda s.)$ in the precondition, which puzzled me for a while (until I realized P:S-¿B).

A: We have written the types of Q and G, and would assume that readers can infer that the precondition of a Hoare triple (abbreviated P by the reviewer) has the same type as the postcondition. The $\lambda\ s.$ should then be clear from the type. A clear indication that predicates depend on states is given before Proposition 2.1.

The notation $(\phi_s t)$ vs. $(\phi t s)$ is super confusing. Pick one and stick to it. Same thing for get/put vs get_x. It's just confusing.

A: It is nevertheless standard in the Dynamical Systems literature.

In Section 5, you mention that "there is one rule per programming construct", but you only show the ones for assignment and differential equations. Do you use the standard ones for the test, sequential composition, star and plus?

A: Yes we do, and we now explain this in more detail.

In Example 5.2 (and all the examples), please define explicitly the hybrid program you're working with.

A: We have made this more explicit.

You claim "increased proof automation" at the end of Section 5 and in your conclusion, but I see no evidence of that. Could you please elaborate? In particular, how do you compare with the proof automation provided by a tool like e.g. KeYmaera X?

A: We do not claim increased proof automation at the end of Section 5. In the conclusion, we now make clear that we use tactics, for instance for standard verification condition generation in Hoare logic. We make no claims about proof automation of the resulting data-level proof obligations. Ultimately, KeYmaera X and our framework depend on interactive theorem proving. It is therefore very difficult to compare proof automation.

Please compare the dR part of your work thoroughly with reference [20], rather than punt with "The relative merits of these approaches remain to be explored."

A: We have explained in the conclusion that Loos' work is not a refinement calculus in the sense of Back and von Wright or Morgan. We acknowledge that Loos' work is prior for us and that we are only the first to give a Morgan-style refinement calculus in the conventional sense. Now we added a few more sentences to compare the two approaches. A more thorough comparison is unfortunately beyond the scope of this paper.

You don't seem to be aware that dL has previously been formalized in both Coq and Isabelle, see the CPP'17 paper: Bohrer, B., Rahli, V., Vukotic, I., Völp, M., & Platzer, A. (2017, January). Formally verified differential dynamic logic. In Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs (pp. 208-221). ACM. In particular, at least the Coq formalization of the CPP'17 paper uses a *deep* embedding while you use a *shallow* embedding. It would be really useful to discuss the respective merits of a deep and of a shallow embedding.

A: We are aware of this work and we have cited it in previous papers. This formalisation is a formal reconstruction and soundness proof of dL in these proof assistants, but not a verification component and in particular not a semantic approach based on a shallow embedding. Nevertheless, we have now added a citation.

Minor: - in abstract, "(Refinement" is weird. Is it a typo?

A: No.

- in abstract, "explains" -¿ "explain"

A: Changed the wording.

- in introduction, "For the latter": it's unclear what you are referring to

A: We think it's OK.