Changelog of Revision

Overview

Firslty, we would like to thank the reviewers for their many detailed comments and suggestions. They were really helpful to produce the much improved version that we submit now.

We carefully followed all the comments and suggestions of all reviewers, and made corresponding changes in our revision.

For the points which had some concrete suggestions/requests and we have made changes to improve our article, they are marked with "DONE" followed by an optional comment about how or where we made the changes. For comments that include questions or doubts, but do not suggest changes in the article, we also tried to respond to them, and hopefully it would make sense. These responds are marked with "ANSWER".

Generally speaking we followed-up on the vast majority of the comments.

The following sections are our point-by-point response to the comments that were raised by the reviewers. Besides these points, we also made changes in various places to improve the overall presentation of our article, but we did not document these changes in this file.

Review 1

Analysis & Related Work

> Clearer explanation of how lAI relates to other work. It would be a service to the field -- and to your readers -- to draw up a feature table comparing the features lAI shares (and does not share) with a variety of systems. In particular...

DONE

We added a feature comparison table at the end of the related work section, featuring some work that we consider the closest to our work, including some of the topics that are suggested by the reviewer.

> The related work section omits two significant areas of inquiry: refinement type systems like F* and Liquid Haskell, and Amin's DOT theory, "DOT with Implicit Functions" from Jeffery, Scala Symposium 2019 Refinement type systems typically have both dependency and subtyping (though there are some issues, e.g., https://github.com/FStarLang/FStar/issues/65). If you'd like to compare to core calculi rather than systems/implementations, maybe take a look at FH, which is the core calculus for Liquid Haskell (Belo et al., ESOP 2011). Belo et al. is related for a less fortunate reason---their reduction substitution lemma was wrong for reasons similar to those mentioned on p26L56, highlighting how critical this property is (Sekiyama et al., TOPLAS 2017 corrects the calculus but doesn't include subtyping). Finally, FH does things the opposite way of most systems: it defines subtyping post facto and shows that it's a semantically safe (if syntactically ill typed)

> optimization.
>
> I also expected to see discussion of DOT (Rompf and Amin, OOPSLA 2016) and some of its followup work (e.g., "DOT with Implicit Functions" from Jeffery, Scala Symposium 2019).

DONE

We now mention DOT (by Amin et al.) and DIF (by Jeffery) and the manifest systems (System lambdaH, FH, FH^sigma) in the related work section.

However we consider the substitution problem mentioned by Sekiyama et al. a different problem from what we mentioned in the discussion (although we call them the same "reduction substitution"). The reduction substitution of FH is a problem in their axioms that causes a contradiction in their system (an ill-formed interaction between convertible types and reduction rules). Our problem of substitution is much "milder" one that limits the design of our operational semantics (we cannot include some reduction rule, because that would cause trouble when proving relevant theorems).

> Clearer indication up front that implicits are necessarily proof irrelevant, and some concrete examples of programs your regime *disallows*.

DONE

We added the following sentence in the introduction:

"Similar to ICC we adopt the restriction that arguments for implicit function types are computationally irrelevant (i.e. they cannot be used in runtime computation)."

We also added a new paragraph and an example in Section 2 called `Computational Irrelevance`.

More general, we believe that the new revision is clearer, in several places, about this restriction.

> A stronger conclusion. Rather than just summarizing the ideas and listing future work, can you give a broader outlook? Supposing you had an algorithmic system... what would you be able to do now?

DONE

We added one more paragraph to the conclusion (at the end) where we describe what we think could be achieved now (if an implementation would be available).

> Are there well formed types that are uninhabited? The monotype restriction on fixpoints may mean that diverging inhabitants can't be found at every type. The question of empty types seems important---on several occasions, the definitions and metatheory are forced to change because we don't know how to find inhabitants for arbitrary types. If there *were* empty types, these metatheoretical concerns would be well justified. But if all types are inhabited, perhaps the theory can be made a touch simpler!

DONE

We rewrote the discussion around the problem of habitability and strengthening, to hopefully clarify why we do not have the strengthening lemma for now, which leads to a series of changes during the generalization of polymorphic subtyping.

Proof Irrelevance

> p13L27 Why doesn't an implicit lambda just erase to its body? I was so confused I went and checked the Coq code, thinking the paper had a typo. It doesn't---I must have an 'understand-o'. But... what am I missing? There's some invariant or idea in the system that I haven't gotten.

DONE

We choose to only eliminate the type annotation, and maintain the syntax structure after elimination (likewise, we leave all the casts in the expression although they practically should not have real runtime impact) for the simplicity of development of metatheory. So that we can find direct correspondence between the reductions of unerased expressions and reductions of erased expressions. The type safety (subtype preservation) is much simpler to reason about this way, than changing the structure of expressions during erasure.

A brief explanation of this idea is added to the paragraph "Deterministic Erased Reduction" in Section 3.2.

> p14L50-57 I struggled with this paragraph. Can lAI handle vectors of implicit length, i.e., `Λn:Nat. λx:Vec n. ...`? A concrete example would really help here.

ANSWER

The "map" function in page 5 (assuming that the arguments are flipped) is of a similar form. So the answer is yes: functions taking Vector-like types with implicit length work. The restriction that we are talking about is regarding *kinds*. Now that the type of the "map" function is at the type level (not the kind level). What we are saying in that paragraph is that implicit polymorphic *kinds* are forbidden. So, if the "..." in `Λn:Nat. λx:Vec n. ...` is a term (like in the map function) then that's fine, but if it is a type, then that's not allowed (since its kind cannot be polymorphic).

We rephrased the paragraph of "Kind Restriction for Universal Types" in Section 3. Added two little examples to illustrate the point and hopefully clarified what the restriction forbids.

> p16L12-20 This paragraph is quite technical, but the gist of it should show up much earlier in the paper.

DONE

We added another paragraph called "Computational Irrelevance" in Section 2.2 that talks about the computational irrelevance of implicit parameters. The related paragraph in Section 3 refers back to the added section.

> p24L40 When you say "Implicit abstractions do not occur in type computation due to the kind restriction of universal types", it would be great to give an example of the kind of program that's forbidden... such an example would go a long way towards making it clear what your system can and can't handle!

DONE

Please refer to the paragraph "Kind Restriction for Universal Types" in Section 3.

> p24L56 I didn't understand this lemma. Shouldn't it be the case that `castdown N` has type `(\x:box. x) *`, and doesn't that type have kind `box`, and doesn't it reduce? What am I missing? (I'm sure it's me, and I just wonder what needs to be said to clear up my confusion. Sadly, stepping through the Coq proofs didn't help me.)

DONE

We added a paragraph under Lemma 22 to explain why such expression is not well-typed.

Syntax, naming, and notation

> p7L17 We haven't yet seen the grammar, so the reader isn't yet equipped to know what counts as a poly- or mono-type in your system (or that the distinction matters, depending on their familiarity with polymorphic subtyping and/or implicits!).

DONE

We added the syntax of types and monotypes to Figure 1, and briefly introduced it in the first paragraph of Section 2.2.

The role of monotypes is mentioned just below in the introduction of the forall-L rule (for the predicativity of relation).

> p9L55 We still haven't seen the grammar... I had to flip forward at this point to be able to know which direction the definition was going in! Relatedly, the use of the word "binder" here to refer to the forall was somewhat confusing.

ANSWER

We rephrased the text in this paragraph significantly. We now show the syntax of the new binder, but we prefer not to introduce the full grammar at this point.

"Binder" should be a standard term to refer to the syntax of a construct that "binds" a variable in the body. Our use of "binder" does not refer to `forall` type itself, it refer to the inhabitants of `forall` types.

> p10L15 It took quite some time to understand that mono-expressions exclude implicit foralls but not explicit pis... and allow *either* kind of lambda. I'm still a bit fuzzy on it, to be honest: a mono-expression can *have* a type with implicits, but it can't *be* one? Right? More clarity here would be very helpful! Some kind of discussion should also go in "Implicit Polymorphism" on p11L24-38.

DONE

We rephrased the paragraph of implicit polymorphism, which first emphasizes the idea of generalization of polymorphic types (and emphasize that the mono-types only excludes forall types not everything else). We also mention the implicit lambda expression at that point.

> p10L42 I'm not sure `Castup` and `Castdn` are the best possible names here. Why not just write **expand** and **reduce** for the terms (or `exp` and `red`)? These would be better rule names, too. The discussion here should perhaps cite Zombie.

ANSWER

We follow the convention of the iso-type systems here. We added a couple of sentences explaining that there are some other cast designs, such as those from Zombie.

> p11L14 I'm not sure a 'mostly' adopted convention is worth mentioning. When *don't* you do that?

ANSWER

For example, in the definition of transitivity, we use e1, e2 and e3. Intuitively transitivity should be a property concerning about subtyping, but our generalized version talks about general terms as well. It is not necessary in this circumstance where e1 e2 and e3 mention terms or types. So the convention is "mostly" adopted instead of "always", we want to emphasize that while e1, e2 or A, B express identical meaning, but there are subtle differences in different contexts where it may matter.

> p12L20 If `Castdn` triggers only one step, why does the outer cast form remain in `R-Castdn`?

DONE

Because castdn triggers type-level reduction, which is performed in the typing rules, while in R-Castdn the castdn operator only serves as an evaluation context. We added a sentence in Section 3.2 to clarify this point.

> p14L20 Maybe call out the three non-structural rules (`S-Forall-L`, `S-Forall-R`, `S-Sub`)? The appearance of `S-Forall` to resolve some issues (p17L51) seems to indicate that "structurality" is an important property!

ANSWER

We are not sure that rules being structural or not is an important property to mention. While the orginal polymorphic subtyping rules are already non-structural, and subsumption rule is very common in a system with subtyping.

> p14L36 Maybe highlight the new kinding premises?

DONE

See below.

> p16L21-25 I assumed the highlighted parts were important when I first read the figure, and I was surprised to see they were in fact redundant. I can imagine several useful highlights here: important kinding premises, premises added due to the issues in Section 4.1.1, places where mono-expressions are required, the three non-structural rules, and redundant premises. I'd put them in that order of importance. Maybe you can draw some `\fbox`en with various forms of dashing, or use colors, or something?

DONE

Now the redundant premises are enclosed in dashed lines instead of being highlighted. And we highlight the newly added premises for rules s-forall-l and s-forall-R, as well as the entire s-forall, point them out in the paragraph "Subtyping Rules for Universal Quantification", and leave the display of other premises unchanged to avoid the figure to be cluttered by too many notations

(kinding restrictions and mono-type restrictions are relatively easy to see).

> p26L24-28 It'd be good to mark some turnstiles with a DK or something---it wasn't immediately obvious that the first turnstile was Dunfield and Krishnaswami's model of Oderskey and Läufer, and the second was your judgment.

DONE

We now specify directly that we prove the subsumption of polymorphic subtyping, based on the subsumption of DK's declarative subtyping. Furthermore, we added DK subscripts to the corresponding turnstiles.

> p26L30-43 I didn't understand these paragraphs at all.

DONE

For 30-33, we expanded the discussion of proofs of the subsumption of the polymorphic subtyping and hopefully it makes more sense.

For 38-43, we rephrased the discussion of the possibility of the reduction of open terms and made it a standalong subsection.

> p28L37-38 "However, we still face..." didn't make any sense to me.

DONE

We rephrased and detailed the said discussion, by specifying that the either of the two choices of instantiation are not more general than the other.

> It would be good to emphasize up front that pi and forall are different---the former explicitly takes an argument and the latter is implicit. I don't believe this is ever actually said in the paper! Calling out the difference between `S-Pi` and `S-Forall` would be a key move (p15).

DONE

We added a sentence in Section 3, "Implicit Polymorphism" to emphasize the difference between \Lambda and \lambda expressions. As for the difference between S-Pi and S-Forall, S-Pi is meant to be a generalization of the standard subtyping rule of function types, and the motivation of S-Forall is explained in Section 4.1.

Coq proofs

> It would be great to relate each theorem in the paper to its name in Coq and the file its proven in---I had to dig around a bit.

DONE

We renamed some of the names in the implementation to better match the name in the paper. Moreover there is a Readme file, that comes with the Coq formalization, that states the correspondence between the theorems in the paper and the theorems in Coq.

> p18L20-23 These lemmas have identical proof scripts. Why can't you prove it in one go? (You'd have to generalize your fancy tactic...)

ANSWER

The problem of left reflexivity and right reflexivity have been different before the introduction of s-forall. The separation of left refl and right refl follows from the original work of unified subtyping, so we prefer to leave them this way.

> p18L49 The substitution theorem here was quite surprising: only monotypes? Looking back at the rules, it's obvious: argument positions demand monotypes anyway, so this property is enough for the system. The text here, though, is confusing: its not *substitution* that "imposes a mono-expression restriction", but rather, the type system itself.

DONE

It was the substitution lemma which imposed the monotype restriction back to the typing rule of s-app. Because of the reason explained under the substitution theorem, because of the subtyping aspect of substitution, and the monotype restriction imposed by s-forall-L, substutition of polytypes does not generally hold. So the flow of monotype restriction is:

s-forall-L => Substitution Theorem => s-app / s-mu

Rephrased the paragraph under substitution to better reflect this logic.

> p20L49 The proof of generalized transitivity seems to go by *strong* induction (i.e., due to the <= on the measures). Might be good to mention this in the paper.

DONE

Added the word "strong".

> p21L11 Might be good to mention that Γ |- τ : A here.

DONE

> p21L34 At the comment about sizes of expressions I immediately wondered why you needed the derivation measure, too---and then saw it a minute later below. Maybe mention up front which cases call for which measures?

DONE

Added high-level explanations of what those measures do when introducing them.

> p22L31-33 The Coq proofs of these progress theorems are more general. Maybe a short paragraph explaining the generality would be helpful?

DONE

We replaced the version of progress with what were proved in the script. Moreover, we added a short paragraph talking about "generalization" very briefly.

> p23L39 I was surprised that the proof of subtype preservation is able to find the valid instantiation (necessarily constructively). I can't see the moment where that's happening in the custom tactics for the Coq proof, though. Some more intuition here would be nice.

DONE

The intuition is explained the "Implicit Instantiation" paragraph under the "Subtype Preservation" theorem, which is rephrased a little bit to hopefully explain clearer.

Other Comments

> p6L13 Maybe say earlier that `MkF` is constructing typeclass dictionaries? It's clear by the end of the page, but it will be easier for readers to know it up front.

DONE

> I would also explicitly mention that typeclass instantiation---a form of implicit argument in Haskell!---is out of scope here, due to proof irrelevance.

DONE

We added a sentence explaining that we have to pass the "typeclass" instance explicitly.

> p7L51 When you say "first attempt", it would be nice to give a clue as to what goes wrong... we find out later that the answer is 'metatheory'. Can you give a short indication that this is the case? Relatedly, are these rules admissible in the final system?

DONE

"A first attempt at direct generalization would be" -> "The idea of a direct generalization is"

We added a paragraph mention the direct generalization is not suitable for formalization for the reasons discussed in later sections.

The admissible rules in the final system are mentioned in subsection "Equivalence to a Simplified System" in Section 4.

> p8L32 Can you explain how Hindley-Milner relates to the ICC rules (i.e., INST at variables and GEN at `let`s)?

DONE

We added some sentences to point out the similar rules (INST and GEN) in HM's declarative system that talks about first rank polymorphic types.

> p17L18 You should indicate up front that this "possible generalization" is incorrect!

DONE

> p17L29 This premise `Γ,x:A |- B : *` just gives us what we would have tried to get by inversion... right?

The purpose is to make the well-formedness `G |- A <: B -> G |- A /\ G |- B` hold, which is not true for rule s-forall-L without this premise.

> p17L43 The core issue here is that the type A may be bottom, i.e., uninhabited... right?

DONE

We rewrote discussion related to the habitability and strengthening.

> p29L54 GHC has type families, which certainly *feel* like type-level lambdas. What do you mean here?

DONE

We expanded the text and discussed type families and how the type functions enabled by type families are not lambdas.

Trivia

> Can you alphabetize your bibliography? The natbib package makes this easy.

DONE

> Some of the papers in the bibliography are miscapitalized, e.g., "System fc" in [15].

DONE

> Many citations are uneven: some just say POPL (e.g., [16]), while others say "ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages". I don't have a strong feeling about which is better, but I think consistency is nice here.

DONE

> Your Coq development works just fine on 8.12.0 (nice work, considering your heavy automation!), though your install instructions are missing the coq-ott package (and the omega tactic is deprecated in favor of lia---a mere warning).

DONE

We added the dependency of coq-ott in the README of implementation repository. And Thanks for the prompt of omega!

> p2L35 "arizes" typo p8L46 "subsumption rule of [the] typing relation" p15 Figure 5 is too tall. p18L54 "take [the] following" p20L10 "head [a] Π type preserve[s] its kind" p22L13 "by employ designs to make" isn't quite grammatical p26L40 "to an open term" -> "on an open term" p26L44 "inside [a] cast operator" p26L50 "is a need" -> "would be a need" p27L16 "we do not really polymorphic kinds" missing word? p27L18 "then this complicates" -> "it would complicate" p27L50 "Such [a] restriction" p27L52 missing space before citation p28L16-19 run-on sentence p28L23 "Π type[s]" p28L36 "problem here stay at first-order" agreement; probably drop the hyphen p28L40 "We consider coming up with" where/when? do you mean it's future work? p29L9 "While since we" extra word? p29L18 missing space before citation p30L25 I'm not certain "hot topic" is the best choice of words here. p31L12 "eliminates [the] typing relation"

DONE

"by employ designs to make" -> "by employing designs that make" "hot topic in research" -> "research frontier"

Recommendations

> Clarify up front that implicit arguments are proof irrelevant---perhaps even in the title! Given the various kinding and occurrence restrictions, what programs do you disallow?

DONE

We did not go as far as the title, but we now mention this in the introduction as well as the overview in more detail.

> A stronger conclusion, with more outlook for practice and/or application. What can we *do* now (or when we have an algorithmic version of your system)?

DONE

> Consider proving whether or not all types are inhabited.

ANSWER We rewrote the discussion around the problem of habitability and strengthening, to hopefully clarify why we do not have the strengthening lemma for now, which leads to a series of changes during the generalization of polymorphic subtyping.

Review 2

Detail Comments

> For what I understand, the idea of unified subtyping has been the main novelty of reference [25], and also casts where already introduced there. Here the main novelty is the integration with polymorphism, however a more detailed comparison with [25] would be useful, also in technical aspects such as choices in the syntax, etc. Also, I had a look at [25] and some aspects (e.g, the role of casts in avoiding to have to compute type equality) are explained much bettere there, perhaps you could import some of these explanations.

DONE

We added one paragraph in related work at the end of "Dependent types and subtyping". There we give more details on the differences between the calculus formalized by us and that formalized in [25].

We also extended the paragraph in overview discussing the addition of cast operators over a conversion rule, hopefully it makes more sense now.

> page 2 41-42 Here it is not clear which is the relation between strong normalization and explicit casts. Could you explain how explicit casts solve the problem? (see comment above, this is explained in [25])

DONE

We added explanantions around the suggested position (P2L41-42) trying to connect the relations among strongly normalization, explicit casts and type checking.

> 17 at this point it is not clear what you mean by "conversion rule"

DONE

We rephrased the paragraph and included a brief explanation of "conversion rule".

> page 5 the example of indexed lists is not completely clear. Is the n:N parameter the initial index of the list? if it is the case can you say this? because the role of indexes is never shown

DONE

We added an explanation after showing the definition that we did not choose a more intuitive example is because it would require GADT encodings, which are considerably more complex (as they involve equality witnesses).

> 29 I do not understand the role of the "r" parameter in the definition of List, can you explain?

DONE

We added a short explanation of the Scott Encoding with an analogy of the continuation passing style.

The parameter `r` is the "final return type" of the CPS functions.

> Please use a uniform font for code everywhere. For instance, map at line 34 is different from map at line 35.

DONE

> line 52: However, you do not show the definition of map

DONE

We rephrased the paragraph to avoid the mentioning of the omitted definition of `map`.

> page 6: In the Functor example, it took me a lot of time to understand what is going on since you do not provide some simple explanation on Haskell-like syntax which I did not remember. It would be enough to recall that at line 13 Functor is defined by a record type, fmap is the name of the field selector, and field selection is written as application of the field selector.

DONE

You are right, the Functor here is a record type that mimic the typeclass. We added explanations to point this out.

> It will also be useful to point out exactly what could not be expressed in, e.g., Haskell. You say (line 33) that F is implicitly instantiated, but F is not Functor Id here?

DONE

We added a sentence to explain with this formulation of Functor, unlike with type classes, the instance has to be explicitly passed as argument.

The parameter `F`, as shown in the type of fmap, refers to the higher-kinded type parameter of `Functor`, in this case, `Id`.

> page 7 you should explain which is the role of the type variables in Gamma; that is, what is only allowed if the type variable is available in Gamma

DONE

We added a sentence to explain the role of \Gamma in section 2.2.

> page 10 line 50: should reference [42] be [41]? (that is, [25])

DONE

> Figure 2: you should at least mention the meaning of the "box" kind

DONE

We added a brief explanation to the role of star and box kind to the first paragraph in Section 3.1.

> page 12: perhaps you should justify better why upcasts are values

DONE

We added the intuitive motivation of why upcasts are values in the call-by-name design of Pure Iso-type System, and a reference to the discussion section about why we did not choose an alternative design.

> Figure 4: to use E both as metavariable and as index of the relation is a very bad choice (I was confused at the beginning)

DONE

We now use \Longrightarrow as the new symbol for erased reductions instead of \longrightarrow_E.

> page 17 line 34 you use "fresh in A" to mean "not occurring in A", right?

DONE

"fresh in A" -> "does not occur in A"

But yes, that's what we meant.

> page 20 line 56 motivated by -> you mean "similar to"?

DONE

"motivated by" -> "inspired by"

> page 24 line 18: which is the "inner" downcast?

DONE

It was a typo and sorry for that. "castdn" -> "castup"

> page 27 line 40: is it the first time that you mention principal type, perhaps should be defined

DONE

We rephrased that paragraph a little and deleted reference of "principal type" to avoid confusion.

Typos and minor comments

> page 2 31-32 repetition: ... a single level of syntax ... using the same syntax page 3 9 it results on -> it results in ? page 5 line 25 ommitted -> omitted page 6 line 38: and instead -> and, instead page 7 50 can be can be -> can be page 8 20 us -> our (?) polymorphism on dependent type language -> something is missing? page 11 line 10 similar syntax -> similar

> line 40: two full stops page 12 line 37 unfol -> unfold page 14 line 47 the -> these page 17 line 56 form -> from page 19 line 19 break -> breaks page 22 line 22 the sense -> in the sense page 23 line 30 a -> an 45 preserves the type of expression -> preserve the type of expressions page 26 line 22 terms type -> terms of type page 27 line 15 "we do not really" should be removed line 20 that, there -> that there line 57 the the -> the page 28 line 23 \Pi type -> \Pi types line 31 depedent -> dependent satisfy -> satisfies Rule rule -> rule stay -> stays Moreover -> Moreover, page 29 line 52 that, -> that page 30 line 28 work of -> work on enforce -> enforces page 31 line 42 of polymorphic -> on polymorphic

DONE

"... using the same syntax" -> rewrite to "i.e. the types are expressed ... using the same syntax as terms" "in the subtyping rule (rule \forall L)" -> "in their subtyping rules (rule \forall L)"

> callcc example: again, please use a different font for code

DONE

References

> [1] American Matematical Society [25] and [41] are the same [14] Guru [15] System FC [20], [35], [55] Haskell

DONE

Review 3

Analysis

> The actual argument for the approach, though, wasn't great. The indexed lists example seems super weird--it appears that indexes increase as the list goes on? But I would expect the index to be the size of the list, and thus to decrease. Consing onto a list won't work once you get to zero, which kind of defeats the purpose...or perhaps I'm not understanding something.

DONE

We added an explanation after showing the definition that we did not choose a more intuitive example is because it would require GADTs (which have significantly more complex encodings and require additional cast support).

> I also didn't understand the encoding technique in the list/map example. What is r in the encoding of List? Is this from Yang and Oliveira?

DONE

We added a short explanation of the Scott Encoding with an analogy of the continuation passing style.

The parameter `r` is the "final return type" of the CPS functions.

> The functor example didn't do anything for me. I expected this to mean functors in ML, but it wasn't. I guess Haskell people will get it, but the example will be obscure for anyone not fully initiated into typed functional programming.

DONE

We have tried to improve the explanation to at least clarify the syntax, following some suggestions from reviewer 2. We admit that the Functor example may not be very intuitive for someone not very familiar with typed functional programming. However, we do think it is a fairly common one and these days Haskell-style Functors are used by programmers in various other languages, like Scala or PureScript.

> My biggest concern is the algorithmics of the system. I understand this is a big challenge and the authors want to stage their work and leave that for another paper. But to me it seems hopeless. I know the Krishnaswami and Dunfield result was very difficult to obtain; at first glance this looks *much* harder. And without automation, I can't see this system doing anyone any good; after all, the whole point of polymorphic subtyping is exactly so that type parameters can be automatically inferred. So there's a gaping whole here--the approach is motivated mostly by practical concerns, but it isn't close to practical until the algorithmic issues are solved. I do think the theory alone has some interest, though; so this is not a fatal flaw, but it is something that decreases my enthusiasm considerably.

ANSWER

We certainly agree that the algorithmic aspects are a big challenge. However we are indeed a bit more hopeful than the reviewer. One reason is that, with our proposal, we can avoid one of the biggest challenges of type-inference work done previously for dependently typed languages: higher-order unification. Because we do not allow implicit type-level computation we can remain in the realm of first-order unification (see the discussion in Section 5.4). Besides that, while we agree that there may be some questions of whether a sound&complete algorithm is possible, we think that, even if that is not feasible, having a sound (but not complete) is a perfectly reasonable, useful and achievable goal.