UNIVERSITY OF WATERLOO

FACULTY OF ENGINEERING | Electrical and Computer Engineering
519-888-4567, ext. 32097 | fax 519-746-3077
ece.uwaterloo.ca

Werner Dietl September 11th, 2024

Review of Vlastimil Dort's Ph.D. thesis:
"Read-only types and purity for DOT"

To whom it may concern,

It was my pleasure to review Vlastimil Dort's Ph.D. thesis: "Read-only types and purity for DOT". The thesis tackles a very fundamental problem in software development: how to handle mutable data. All popular programming languages provide a way to change the state of the program by manipulating the heap. In object-oriented programming languages, this can create an intricate web of references between objects and all references to an object can be used to modify that object. Such in-place updates allow performant computations, but unexpected modifications can lead to hard to understand errors, both in single-threaded and in particular also for the modern multi-threaded world.
Read-only types enable the developer to make a distinction between a reference that can be used to modify an object and a safer kind of reference that can only be used to read values from an object. The purity of a method determines whether the method can side effect the heap during execution or not. Both concepts have been studied extensively over the years, but no general-purpose language has integrated these concepts.

The main novelty of this Ph.D. thesis is an encoding of read-only types using the existing language mechanisms of the Scala programming language, in particular, path-dependent types and intersections and unions of types. By directly encoding read-only types using existing language mechanisms, the programming language does not need to be changed to support this additional concept. This is a major scientific insight that could propel the concept of read-only types into a mainstream programming language.
Besides an informal description of the system, the thesis also presents a mechanized formalization of the type system, called roDOT, and a proof that the Immutability guarantee actually holds for the formalized language. Using mechanized proof systems is still very challenging and achieving this level of formalization is impressive.

The second novelty of this Ph.D. thesis is a formalization of pure methods, building on the earlier formalization of read-only references. This extended system required further formalizations of the semantics of the system, culminating in the proof of a Side-Effect Freedom property. This system is again formalized using the mechanized formalization system.

The third novelty of this Ph.D. thesis is the formalization of a transformation guarantee that shows that calls to side-effect free methods can safely be reordered in application code. This property can be useful to show the independence of method calls.

The thesis goes beyond a formal treatment of the concepts by also implementing the formalized concepts and performing case studies of their usability. The extension to the dotty Scala compiler show that the language extension is practically applicable and useful.

As motivated earlier in this review, handling mutability is an important problem and this thesis makes significant, novel contributions to this research area. The insights can enable safer programs, support program verification, and increase the reliability of multi-threaded applications.

The thesis is structured as follows. Chapter 1 gives a general overview of the research problem and setting. Chapter 2 provides an extensive review of the required background, in particular, of the different Dependent Object Types (DOT) formalizations of Scala. Chapter 3 then presents roDOT, the novel extensions of DOT with read-only types. This chapter manages to present very complex, technical material in an understandable way, very nicely complementing the mechanized formalization. Chapter 4 then formalizes the side-effect freedom property and expresses it from three different viewpoints and then presents the transformation guarantee. Chapter 5 discusses the implementation and experiments. Finally, Chapter 6 concludes and discusses possible future work.

I have a few questions for discussion in the defense, in particular: 1) What is the intended use case for read-only references in Scala? 2) What are trade-offs between encoding read-only references using unions/intersections vs. annotations? And 3) What further transformations besides method call swaps could be built? I include a list of small typographic errors and detailed questions below. This list is meant as hopefully useful feedback and does in no way diminish the tremendous contributions of this thesis.

Overall, the thesis is very well written and clearly articulates the intuition behind the formalizations, which is very often ignored by researchers. The author is clearly able to solve a very challenging scientific problem in a creative way, formalize the solution, and present the results in an understandable way. The successful publication of several papers also speaks to the high quality of the presented material.

In my opinion, Vlastimil Dort clearly exceeds the high bar to be awarded a Ph.D. degree.

Please do let me know if you have any questions.

Best regards,

About the letter writer: Werner Dietl is an Associate Professor in Electrical and Computer Engineering at the University of Waterloo. He is a researcher in safe and productive software development whose theoretical results and practical tools are helping developers create high-quality, trustworthy code for embedded, desktop, and server applications. He has achieved exceptional technology transfer success through work on the specification and implementation of type annotations now used in Java 8, positively impacting ~10 million Java developers worldwide. Multi-national companies, including Amazon, Google, and Uber, use Prof. Dietl's pluggable properties research daily, particularly the EISOP Checker Framework.

In 2019 Prof. Dietl received an Early Researcher Award from the Government of Ontario and he held a Google Visiting Faculty position in 2017, earning him an intensive seven-month internship at the Google headquarters in California. Before joining the University of Waterloo in 2013, he worked with Prof. Michael D. Ernst and the SE.CS and WASP research groups at Computer Science & Engineering, University of Washington, and he was a research and teaching assistant at the Chair of Programming Methodology, ETH Zurich, working on his doctoral thesis under the supervision of Prof. Peter Müller.

For more details, see his homepage at https://ece.uwaterloo.ca/~wdietl/

Detailed list of comments (pages refer to PDF page numbers):

Page 14:

- “parties including researches.” Should be “researchers” or something else.
- “When a extension” should use “an”.
- “recently explicitly nullable types, safe references” citations for these two proposals would have been nice.
- “formalization, In particular” incorrect comma or capitalization.
- “most popular languages…” C++ should probably also be on that list. Some citation for the used popularity ranking would have been good.
- Throughout, it should be “object-oriented languages” and “run-time system”, etc. That is, use a hyphen if the words together form an adjective to some other word. See e.g. https://jamesrwilcox.com/runtime.html
- “with Java includes” odd transition.
- “from Functional programming” odd capitalization.
- “Withing the space” typo.
- “dynamically typed lang” I would write out “language”. And it would be “dynamically-typed language”.

Page 15:

- “In the simple view” odd wording.
- “Java and Scala are not sound” A citation of that paper would be good.

Page 16:

- “It has, however, been shown that” Missing same citation, if the same unsoundness is meant.
- “using various trick requiring” should be “tricks”.

Page 17:

- “Type members are an o-o analogue of generic type parameters.” In several places here, there would be papers you could cite that introduced these concepts.
- “do not belong tho the same” typo.

Page 18:

- Figure 1.2 does not say what specific run-time error is thrown, which at least for the Nothing example I don't know.
- "un-interesting pieces of code…" An example of this would have been interesting.
- "If the unsoundness is not well understood…" An example of the errors and wrong code would have been nice.
- In Section 1.4 an introduction to soundness proofs of programming languages would have been nice somewhere – from basic pen&paper syntactic soundness to uses of Coq/Isabelle. This jumps into mechanization, without citing the foundations.
- "It is common to mechanize" It would have been nice to cite alternatives people use. I'm also curious how common this is.

Page 21:

- The overview paragraph doesn't mention Section 2.3.
- "just one its supertypes" odd wording.
- "? extends String" String is a final class, so using it as a bound is odd.
- "because the allow" typo, should be "they".
- "but in Compositional Programming disjoint" Is Compositional Programming introduced in [86]? Some of the wording here reads oddly.

Page 22:

- The Top vs Bottom discussion could have been more symmetric, to make the distinction easier to see.
- Nit: Many of your dashes look like "--" instead of the proper "---", e.g. see Tip 2 in https://latex-tutorial.com/tips-for-professional-latex-typesetting/
- "Type member" and "overridend" typos twice in the example.

Page 23:

- "type members have a lot in common with generic type parameters" Some citation?
- "Refinement types" Some citation would have been nice here, in particular a distinction to the Refinement Types ala LiquidHaskell would have been nice.
- "Type annotations" citations to the corresponding language manuals would have been nice.
- "can contain an arbitrary value" Is it a value or an AST? An example would have been nice.

Page 24:

- “Checker Framework” with upper-case “F”. The transition to Java could have been called out. Calling nullness a “simple” type system is a big understatement, given how much work continues to go into that type system.
- In this discussion of refinement types and the CF, a mention of Property Types (https://dl.acm.org/doi/abs/10.1145/3485520) would have been good.
- “decidability, typing algorithms” Missing “and” or other wording.
- “language extensions [102]” there are so many, listing a few more would have been nice.
- “an earlier versions of Scala” Singular/Plural mismatch.

Page 25:

- “variable names in parenthesis, which we use for variables.” Unclear wording.
- “all variables are term-level” Unclear what that means.
- “to convert a element” Convert sounds like something is happening to the element. “a” should be “an”.

Page 26:

- “is more precise” sounds like T1 is always more precise. Usually subtyping is reflexive.
- “Typing rules” vs the earlier “Typing judgment” is unclear. Could they be merged? Or their contrast highlighted.
- “rules that have not judgment premises” wording.
- “Program (t)” uses the same “t” as Term – that should be highlighted.
- (d) lists “a field, a method or a type member”, whereas (D) uses a different order. Making this symmetric would be good.
- “it is defined with type” Unclear what that means here. “as a mutually inductive definition” with what?
- “The heap stores the object that can be…” Plural “objects”?
- “An answer” A better introduction to “answer” would have been nice. Answer to which question?

Page 27:

- In “Semantics” and/or “Small-step semantics” citations for alternatives would have been nice.
- “never get stuck” uses the wrong quotes.
- “Judgment” vs the earlier “Typing judgment” and “Typing rules”??

- “Decidable typing” “if every instance of the typing judgment” Is it every instance of the judgment? Unclear what “instance” means here.
- “is conjectured” citation would be nice.
- “discoveries of unsoundness” citation.
- “feature scan be modeled” spacing between words.
- “is the proof soundness” wording.
- “and the type checking implemented in the compiler” The proof of DOT doesn’t tell you anything about what dotty implemented, unless it is automatically generated from DOT.

Page 28:

- “is that can be extended” wording.
- “proof of such extension will” wording.
- “versions of dot” capitalization.

Page 29:

- “object filed be” typo.
- “made several version” plural.
- “A version with mutable fields but without constructors… A simplified version [63] …” that sounds like the same system? Something here is off.
- “based on the type of the desired type” odd wording?
- “a new possibilities” wording.

Page 30:

- “If was designed” wording.
- “System D Square [14]” missing period.
- “Restricted versions of D<: has been designed, that has” wording.

Page 31:

- with empty stack, heap, etc.” What is the “etc.” for?
- “focus of execution” I think this was the first this term was used for the “term” in a configuration.
- “are term-level.” This is never explained.

Page 32:

- Rule BTT-Fn uses “z” instead of “z_2”.

Page 33:

- “are in A-normal form (ANF).” Citation would be nice.
- “written to a field, An apply” separator.
- Example 1 uses “x_2.a_1”, but then talks about a_2.

Page 34:

- “Programs in kDOT and the baseline DOT” At some point I lost track what is meant with kDOT and baseline DOT… and there were multiple versions of kDOT...
- “a argument” typo.
- “stores the value of x_m in field a” Should that be “x_a”?

Page 35:

- “by type selection” I don’t think we saw how to select these?
- “Subtyping rules are in Figure 2.4.” The order of the discussion or figures was a bit odd here.

Page 36:

- Some of the rules in Figure 2.7 could have used more explanation – their basic shape seems inconsistent.

Page 37:

- “Invertible Typing” the typing judgments suddenly use ρ, which hasn’t been introduced yet. There needs to be an introduction and forward reference.
- “to prove Lemma 22.” This again needs some explanation and forward reference to that lemma.
- “and a mutability declaration”. Also out of place here.

Page 38:

- “that there all proof” wording.
- “Th basic” wording.

Page 39:

- “several several” wording.
- “locally-nameless representation” isn’t introduced until next page.
- “and types a re formed” wording.

Page 40:

- “definitions represented by” wording.

- “definition consists constructors” wording.
- “uses a with unified” wording.
- “locally nameless notation[33]” should be “locally-nameless notation [33]”
- “indices [41].” spacing.
- “and so on..” punctuation.
- “The syntax, nor” missing “neither”.
- “which larger” wording.
- “closing operation” wrong quotes and odd wording.

Page 41:

- “All is needed is” wording.
- “prove properties Transform and Collect for” these properties haven’t been introduced.
- “defined an proven” wording.
- “The Coq proof assistant is based…” Found it a bit odd to see this paragraph here, after so much Coq already used.

Page 42:

- - subsumption, opening terms.” dash and wording.
- “the filed assignment” wording.
- Some file lists should be separated by commas.
- “Correspondence of heap with” wording.

Page 43:

- “inversion to be practically useless.” An explanation why would be good.

Page 47:

- “are either deeply or shallowly immutable” This is the only place that uses the words “deeply” and “shallowly”.
- Here, or somewhere else, I would have expected a citation of “C++ const and immutability: An empirical study of writes-through-const” ECOOP 2016 or of Eyolfson’s Ph.D. thesis.

Page 48:

- Mutability polymorphism. The difference between this and viewpoint adaptation does not become clear here, because it again refers to viewpoint adaptation.

Page 49:

- “There can only be one polymorphic qualifier” Why? Once can either introduce multiple qualifiers or use one annotation with a name/value to differentiate.
- “combine qualifiers from multiple declarations” Not clear what this would mean.
- “Them mutability polymorphism…” wording.

Page 50:

- The “Union Types” section needs a clearer conclusion.

Page 51:

- “as as” typo.
- “In polymorphic methods, we type it as read-only” This is again confusing the distinction between a polymorphic method and viewpoint adaptation.

Page 53:

- “The syntax of roDOT is defined in Figure 3.2; the shading highlights changes from the baseline DOT (Figure 2.1).” There is no shading!

Page 58:

- The use of “ρ” in the static rules is surprising. Why is this needed?
- In ST-Met I am surprised why the value-parameter type is in the environment for the receiver parameter subtype test.

Page 63:

- All the rules have an “s: T_4” in the environment, which doesn’t make sense to me for most rules.
- “stack s” should be σ.

Page 64:

- There should be a reference to Figure 3.10 in the text.
- “made about pes” wording.

Page 66:

- Like on page 63, I’m confused by the “y/s : T_4” in all environments. That should again be explained.
- There is no reference to Figure 3.12 in the text.

Page 68:

- In Figure 3.15: “lemmata” is plural and shouldn’t be used for a single lemma. Some of the boxes contain no reference to a lemma/theorem/definition.

Page 69:

- “(not shown)” Doesn’t that figure contain all? Why not?
- “then there exists w_2” shouldn’t that also list T_2 and \rho_2?

Page 71:

- Theorem 8 uses an unbound \Gamma_1. Should the \tau_1 be \Gamma_1?

Page 74:

- “in some cofiguration” typo.

Page 75/76/78:

- References to Figure 3.19. should actually be to Figure 3.19 for the approximations.

Page 80:

- “DOT by Ifaz Kabir” cite.
- “Finally, Section 3.6” that’s the current section!
- “Definitions of an object represented” wording.
- “Hoverer” typo.
- “are involve” wording.

Page 81:

- “a an additional” wording.

Page 84:

- “support for support” wording.
- “In Rust, mutability is tied to ownership.” Some citations here? There are e.g. separation logic systems for Rust.
- In general, some citations for separation logic, fractional permissions, and program verification would have been good, as these domains solve many similar probems.

Page 85:

- IGJ came before ReIm, so I would reorder these paragraphs.
- “for the C# language [52]” Is this feature actually part of C#? Or is this just one research proposal?
- A citation of the “Immutability” survey paper would also be good. https://link.springer.com/chapter/10.1007/978-3-642-36946-9_9

Page 89:

- A few more citations would have been good here – where is Pure first introduced? What existing annotations exist, e.g. in JML and CF.
- “are designed behave like” wording.
- “[56] end eliminating” should be “and”.

Page 90:

- “based on new definition” wording.

Page 91:

- “In section Section 4.7” duplication.
- “are actually designed as side-effect-free and meant to work like pure mathematical functions, producing the same result on each invocation.” This mixes SEF and determinism. Would be nice to already separate these clearly.
- “The idea that a method is pure can be expressed…” A citation for JML would be good here, as that’s one long-standing definition of purity and more fine-grained side-effect specifications come from. “JML Reference Manual” by G. T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. Cok, P. Müller, J. Kiniry, P. Chalin, D. M. Zimmerman, and W. Dietl. June 2008. Available from http://www.jmlspecs.org/.

Page 92:

- Some mention of “observational purity” would be good here already.
- “The following sections each…” No mention of 4.2.5.
- “are modifications of global state or IO operations”. What is the global state? Just the heap? Or something else, too?

Page 93:

- “3. Termination” Some discussion here and references to tools/papers on termination would have been nice.
- “The discussing in the following… we will only SEF property.” This whole paragraph needs rewriting.

Page 94:

- The ordering of the three perspectives was a bit odd, in particular as the discussion is then in a different order.
- “view from perspective is described” wording.
- “The heap can only be modified…” somewhere here I would have expected a mention of other method calls and what has to hold for them.

Page 95:

- “informal statement of 7” Not clear what “7” refers to.
- “Definition (2, static SEF condition..” Again, the “2” is a bit out of place. Same with a few other definitions in this chapter.
- Example 5 ends with “which may mutate the heap.” This mutation is limited to the newly created object, right? It would be nice to call that out and to highlight that the new object cannot have a mutable reference to an pre-existing object.

Page 96:

- “We will revise the way to recognize…” a reminder why this is needed would be good here.

Page 97:

- In the discussion of the call-swapping transformation, I would have expected some mention of independence of the arguments to the methods. E.g. “x = y.m(); z = y.m2(x);” obviously can’t be swapped, even though they are both pure method calls.
- “can transformed” wording.

Page 98:

- “to be Turing-complete [107] language” wording.
- “so conjecture that” wording.
- “do not stat type-system guarantees” typo.
- “we identify issues” wording of whole sentence.
- “which were proven by hand for the original roDOT [43]”. In several places in this discussion I would have expected a clear relation to Chapter 3.
- “and discuss the meaning” odd paragraph start.

Page 100:

- “In Definition 2 “A method is statically SEF” should specifically say “A method m”.

Page 101:

- “and one modified typing rule” … “The typing rule TT-Call subtying rule ST-Met” Should it be two changed rules? Something doesn’t align with Figure 4.1.

Page 102:

- “is read-only. We can use it improve” wording and sentences.
- “on the heap based, on the types” odd comma.

Page 104:

- The proof of Lemma 2 in DOT… Should this to the just introduced Lemma 24?

Page 107:

- Figure 4.4: Rule LTA-Sub looks odd.

Page 109:

- “declaration type type by” wording.

Page 110:

- Proof sketches for Lemmata 27 and 28 are missing.

Page 111:

- “has to to go” typo.
- “Lemma 35… 2.5in” Some LaTeX slipped in?

Page 115:

- Definition 7 uses “prefix of \Sigma_3” whereas Definition 8 uses \Sigma_1 \ subseteq \Sigma_2” even though both mean the same. Introduce the notation first and use it consistently.
- Definition 9 uses “\forall y \in dom \Sigma” Which \Sigma?
- “it forbids modifications of existing objects.” Should this be “new objects”? All definitions forbid the modification of existing objects? Or is the point modifications that are restored to their original values?

Page 116:

- Theorem 37: “which is a read-only methods” wording.
- “contains the all the items” wording.
- “or c_1 |-> …” should this be c_1 |→^k?

Page 117:

- “was proven for roDOT [43]” again, relation to Chapter 3 should be made clear.
- “and is included of our mechanization” wording.
- “only until the end” wording.
- “The rest of this section” should be clarified. Also the gray border could be mentioned.
- “location and references” mismatch.
- “may use different variable” wording.
- “to be able treat” wording.

Page 118:

- Definition 10 “on the left… on the right” Unclear what this refers to… the elements of the pair? And in what is something renamed??
- Definition 11 uses “i” subscript for substitutions, which also isn’t clear.

Page 119:

- “fact that new in all” wording.
- Lemma 39, the text mentions “to a similar configuration, but I don’t see a c_1’ \approx= c_2’ in the formular.

Page 120:

- “Heap correspondence:” Description ends with a “:”.

Page 121:

- In Lemma 44 I miss a binding for W and Y.

Page 122:

- In 4.5 “the code x1.m1(); x2.m2();” that is dead code if m1/m2 are SEF. You could discuss that.
- “calls, but build” wording.

Page 123:

- “because of the following points.” This sounds like a new list is starting, but it is part of the ongoing bullet points.

Page 124:

- “location names can different.” wording.

Page 125:

- “A heap transformation is… “ uses \sigma instead of \Sigma. End of paragraph it uses the words Gamma and Sigma.

Page 127:

- “contain premises… these definition … but applying to two…” wording.

Page 131:

- “then moves to the stack” wording.

Page 132:

- In Figure 4.15 I would move the “*” so that it doesn’t look like it’s part of the nodes. It’s a label on the edges, right?

Page 133:

- In Definition 21, both calls have the same type T. Why? Isn't T the type of "t"?
- "need to to prove" typo.

Page 137:

- "is extended with all from" wording.

Page 139:

- "are Haskell, Coq" Separate by "and". Citations would be nice, ideally to their purity.
- Similarly, some citation for the pure sub-language of C++ would have been good.
- "Purity annotations. Several programming languages…" some citations would have been good.
- A discussion of JML is again missing.

Page 140:

- "the programming language differ" wording.
- "Such method cannot return" wording.
- Some citations in 4.7.1 would have been nice.
- "one update field" wording.

Page 141:

- "has the the bottom" typo.
- "for general programming, It is," typo.
- For Java, discuss distinction "final" vs immutability and purity.
- "do not invalidate flow-sensitive types of local variables." This should be fields, at least in addition to local variables – a method cannot modify the local variables of a different stackframe. For most type systems (nullness, index) this cannot invalidate the value in a local variable. For some "deeper" type systems, locals also need to be invalidated.
- Throughout, use "the Checker Framework", it reads nicer.

Page 142:

- "This type system only SEF property" wording.

Page 143:

- For C++, discuss "const" and the earlier paper on C++ const I referenced.
- "calls to such method" use "a method" or "methods".
- A citation for the pure sub-languages of C++ would again be nice.

- For Dafny, a clearer statement about what “pure” means would have been nice. Also, what about determinism?

Page 144:

- JML allows fine-grained read/write effects
- Separation Logic and Region Logics should be mentioned.
- Permission systems, e.g. Viper should be cited.

Page 145:

- “with experimental implementation a” wording.
- “same time a demonstrated” wording.

Page 146:

- “when it’s type is required” “its”?
- “into a TASTY format” Are there multiple TASTY formats? What is it?
- “or modify the tree,” sentence not ended correctly.
- “not String, but x.type” Should that be “s.type”? Or literally “x”.type?

Page 149:

- “a class refers the containing object” wording.
- Some of the discussion between “@Mutating” and “@ReadOnly” got confusing and the examples aren’t discussed.

Page 151:

- “the same way as with viewpoint adaptation” VP can only be used to combine one receiver type with some declared type. Allowing arbitrary unions sounds much more flexible.
- “would required adjusting” wording.
- At least Java has method type argument inference, “diamond” inference, and auto. Not inferring parameter and return types is intentional for readability.
- “reduces Annotation overhead” capitalization.
- “Because mutability integrated” wording.

Page 152:

- “pieces of code shows” wording.

Page 154:

- “when the methods is” wording.
- “that Although” capitalization.

- “Type system regularity:” To make it regular with all other bullet points, end with “.”.

Page 155:

- “an immutable collection are immutable” wording.
- “the collection type C and…” I would list them in the order as in the code.
- I wish there were more discussion and an example of higher-kinded VP.
- “Here we assume that toString is declared as read-only.” But toString is not used in the example. Should it be? I don’t get the connection to the code.

Page 156:

- I wish there were more discussion of Element Access Methods. I think there is a nicer alternative design.
- “because on that will” wording.

Page 157:

- “of the collection a and return a collection” wording.
- “The predicates are typically assumed” What does this mean? It is not enforced by the type system?

Page 158:

- “with viewpoint-adapted element types.” An explicit statement how much of this the formalization roDOT can handle would have been nice.

Page 160:

- “borrowing and fractional permissions” More citations!
- “in the DOT caluculs. where the” wording.

References:

- Some citations are very long, others very abbreviated.
- [45]. Michael D. Ernst isn’t the only author of Pure.
- [57] vs [58] Different authors?

Page 181

- “is on the track” wording.