Prof. Dr. Philipp Haller
KTH Royal Institute of Technology, Sweden
School of Electrical Engineering and Computer Science (EECS)
Division of Theoretical Computer Science (TCS)

Stockholm, September 4th, 2024

Expert Report on Mr. Dort's Dissertation Entitled "Read-only Types and Purity for DOT"

It is my pleasure to provide an expert report on the PhD thesis report by Vlastimil Dort. The dissertation presents several original contributions in programming languages research concerned with safe, typed object-oriented programming. The focus of the thesis is on controlling side effects in object-oriented programming languages that support mutable objects, that is, objects with mutable fields. By restricting side effects using static types, the ability to reason about programs and their reliability can be increased significantly. Scala is a modern programming language with a flexible and expressive type system, grounded in a foundational family of calculi (the DOT calculi) for which type soundness has been established formally. While Scala and its type system have excellent support for functional programming where side effects are either strictly limited (for instance, to local state within a function, say) or absent entirely ("pure functional programming"), to date it has been impossible to specify and enforce side-effect freedom. As a result, the ability to reason about such mostly-functional code has been fundamentally limited. The present doctoral dissertation of Mr. Dort proposes extensions to Scala's type system, formalized in the roDOT calculus, a variant of a core calculus in the DOT family, which enable expressing **reference mutability**, that is, whether a reference is either mutable (i.e., unrestricted) or read-only, preventing the mutation of an object's fields through that reference, transitively. These type-based extensions enable, for the first time, to specify and enforce a fine-grained notion of side-effect freedom both in DOT-based calculi (formally) and in Scala (practically).

In the following we summarize contributions and key parts of the content of the main chapters. These summaries serve to inform the assessment of the dissertation as based on the technical content of the dissertation. Chapters which are not mentioned explicitly are not ignored, however. Instead, their contribution is reflected in the overall assessment provided at the end of this report which also evaluates and assesses orthogonal aspects of the presented work.

The PhD thesis makes **three main contributions**.

The **first contribution** is roDOT, an extension of a core calculus in the DOT family, which provides type system extensions for expressing and enforcing reference mutability. The design of roDOT is unique in the way in which features of DOT are utilized in order to keep the required extensions as economical as possible. The meta-theory consists of two main results: first, type safety; second, an immutability guarantee, which expresses an essential property provided by the reference mutability in roDOT. Type safety states that for a well-typed term, reduction terminates in a finite number of steps or reduction continues indefinitely, i.e., the reduction of a well-typed term does not result in a stuck term/configuration. The immutability guarantee states that in a well-typed configuration, an object is either immutable (i.e., it is not mutated after any number of steps) or it is/was reachable by mutable references. This guarantee ensures that the only way to mutate an object is by using a mutable reference. In order to state this guarantee precisely, a notion of **mutable reachability** is formalized. On object is mutably reachable if it is reachable in the current configuration using only read-write references. The theorem providing the immutability guarantee states that each object in a typed configuration is either mutably reachable, and can therefore be mutated, or it remains unchanged after any number of reduction steps. The roDOT calculus and its meta-theory have been mechanized using the Coq proof assistant, including the proofs of all theorems. Therefore, we can be confident in the correctness of the presented results. The mechanization is the result of a collaboration with Yufeng Li from the University of Waterloo.

The roDOT calculus and its meta-theory **can be regarded as a new scientific result**, since it is the first result to, in the context of a DOT calculus with mutable fields, (a) formalize reference mutability and (b) establish an immutability guarantee. **The importance of this result is highly significant for programming languages combining functional and object-oriented programming**, since it shows how reference mutability can be added to the DOT family of calculi in a sound way, providing an essential immutability guarantee. The immutability guarantee is, in turn, important to guarantee other properties, including a side-effect freedom guarantee (see the next contribution) and, potentially, a data-race freedom guarantee. A **key possible application of the result** is to serve as a foundation for a principled and sound implementation of reference mutability for the Scala programming language.

The **second contribution** of the PhD thesis is an extension of the above roDOT calculus and type system with the concept of side-effect freedom (SEF). Being able to know which methods are guaranteed to be side-effect-free eases reasoning about program behavior and enables various program optimizations. An **important application** of the SEF guarantee is **the safe transformation of programs invoking SEF methods without changing their behavior**. Desired safe transformations include caching method call results, and reordering calls to SEF methods **for the purpose of optimization**. The thesis formalizes the SEF guarantee in the context of an extended roDOT calculus, and provides the proofs of several key theorems: type safety; the SEF guarantee (which establishes that calling a SEF method does not modify existing objects in the heap); and, finally, the transformation guarantee (which establishes that two programs with swapped calls to SEF methods produce the same results). The extended roDOT calculus and its meta-theory have been mechanized using the Coq proof assistant, including the proofs of all theorems. Therefore, we can be confident in the correctness of the presented results. The mechanization is the result of a collaboration with Yufeng Li from the University of Waterloo.

The extended roDOT calculus and its meta-theory **can be regarded as a new scientific result**, since it is the first result to consider side-effect freedom (SEF) in the context of a DOT calculus with mutable fields. **The importance of this result is highly significant for programming languages combining functional and object-oriented programming**, since the SEF guarantee (together with the theorem on safe program transformation) enables important program optimizations for such languages. A **possible application of the result** would be provably sound program optimizations performed by compilers for the Scala language.

The **third contribution** of the PhD thesis is a prototype implementation of the theoretical results in the reference compiler for the Scala language, Dotty. Moreover, a case study is presented which applies the prototype implementation to Scala's collection library, the most important part of Scala's core library. The case study exposes patterns (concerning required type annotations) that result from adopting the proposed extensions for reference mutability. The case study **can be regarded as a new scientific result**, since no such study and analysis has been performed previously. **This result is very important for the development of languages like Scala**, since it shows in which way generic collections libraries would have to be evolved in order to benefit from the presented approach to reference mutability. A **possible application of the result** is a sound and production-ready implementation of reference mutability for the Scala language.

In summary, the thesis presents **several new scientific results of high importance to research in the area of programming languages**, with a particular focus on languages combining functional and object-oriented features, such as Scala. The **potential applications of these results are highly promising**, since they would enable powerful guarantees, such as an immutability guarantee and a SEF guarantee, to programs written in Scala, a language that is used in a number of large-scale, business-critical systems.

The extent, length, and form of the dissertation are adequate. Moreover, the text of the dissertation is of excellent quality regarding language and style. Figures, listings, and examples are very well thought-out, readable, and help the illustration. In particular, I appreciate the annotations on the sides of theorems and lemmas which translate the formal language into informal text to help guide the reader.

The dissertation is based on several prior publications with co-authors. There are no indications that the presented scientific contributions have not been achieved independently by Mr. Dort. Furthermore, there are no indications that rules of good scientific conduct have been violated.

In summary, **the thesis clearly proves the author's ability for creative scientific work**. I consider the dissertation of Mr. Dort to be an excellent dissertation on the international level, and therefore recommend acceptance.

<u>Questions to discuss at the defense:</u>

Q1: Page 52, Figure 3.6: In typing rule TT-Write, why does x1 have to have type T1? Wouldn't it be less restrictive and still sound if x1 would be required to have type T2?

Q2: Could the transformation framework presented in Chapter 4 potentially be adapted/extended to ensure deterministic concurrent execution (assuming the calculus would be extended with concurrent tasks)? Ensuring deterministic program results in the presence of non-deterministic interleavings of concurrent tasks, usually requires reasoning about permissible reorderings of statements/expressions/tasks.

Q3: What do you see as the main challenges for the adoption of your approach in the mainline Dotty compiler and Scala's collections library? To reduce the amount of type annotations, could certain abbreviations or defaults (thus avoiding annotations) perhaps be helpful?

Yours sincerely,

Prof. Dr. Philipp Haller
KTH Royal Institute of Technology, Sweden
School of Electrical Engineering and Computer Science (EECS)
Division of Theoretical Computer Science (TCS)
Email: phaller@kth.se, Phone: +46 8 790 81 20