Mainstream object-oriented programming languages, such as Java or Scala, typically allow objects to be mutated by assigning new values to their fields, but it is also common to write code that only accesses objects in a read-only way. Reference mutability is a technique for controlling mutation by distinguishing read-only and mutable references with types. It has been thoroughly studied in Java, and implemented as compiler extensions. Scala is an evolving programming language which integrates many advanced type system features, most notably path-dependent types. To address questions about soundness, the formal Dependent Object Types (DOT) calculus has been developed, which provides a formal proof of soundness for the core of Scala's type system.

In this thesis, we explore the possibility of using DOT's features to integrate reference mutability. We define the roDOT calculus, which is based on a version of DOT with mutable fields, and encodes the mutability of object references using a special type member. This encoding makes it possible to use path dependent types to refer to mutability of a reference, and use intersection and union types to combine mutabilities and implement viewpoint adaptation, ensuring the transitive property of read-only references. In addition to updating the type soundness proof of DOT to this extension, we state and prove the Immutability guarantee, which formally states that objects in roDOT can be only mutated through the use of mutable references.

Modern code also often makes use of pure methods, which have properties of mathematical functions. The ReIm type extension for Java showed the connection of reference mutability to purity of methods, in particular, the side-effect-free property. We explore purity conditions in roDOT and pose the SEF guarantee, by which the type system guarantees that methods that can be typed with read-only parameters are side-effect free. Applying the ideas of ReIm to roDOT required just a few changes to the type system, but necessitated re-working a significant part of the soundness proof. We proved the SEF guarantee by applying the previously stated Immutability guarantee.

In addition to the SEF guarantee, we state a transformation guarantee, which ensures that in a roDOT program, calls to SEF methods can be safely reordered without changing the outcome of the program. The transformation guarantee is proven by applying the SEF guarantee within a framework for reasoning about safe transformations of roDOT programs.

We mechanized the definition of roDOT, the soundness proof and all the guarantees using the Coq proof assistant. As a demonstration of the possibility of bringing the ideas of roDOT to Scala, we provide a patch to the Dotty compiler, which implements basic reference mutability checking.