

Functional languages such as Haskell and F# make code more testable, inherently thread safe and easier to reason about through their functional purity. However, the adoption of immutable data structures introduces efficiency costs, as many operations require copying rather than performing in-place modifications. To address this, the Koka language introduced a novel mechanism known as fully-in-place functional programming (FIP), which enables safe in-place updates while minimizing unnecessary memory allocations. Nonetheless, Koka's garbage collection and extensive feature set complicate the task of isolating FIP's specific memory efficiency advantages. The goal of this thesis is to design a minimal functional language called StaFip that supports fully in-place updates utilizing the FIP calculus and omitting garbage collection. This will allow for a comparison of the performance of the FIP approach against that of a conventional implementation for algorithms such as quicksort, red-black tree insertions, and finger tree insertions. The findings of this thesis demonstrate that a language employing the FIP calculus in a garbage collection-free environment can achieve a significant increase in performance and memory efficiency.