UNIVERSITY OF AMSTERDAM

Efficient Immutable Collections.
M.J. Steindorfer

This thesis proposes novel and efficient data structures, suitable for immutable collection libraries, that carefully balance memory footprint and runtime performance of operations, and are aware of constraints and platform co-design challenges on the Java Virtual Machine (JVM). Collection data structures that are contained in standard libraries of programming languages are popular amongst programmers. Almost all programs make use of collections. Therefore optimizing collections implies automatically increasing the performance of many programs.

Today's collection data structures are mostly one-off solutions. This is problematic, since statically encoding data structure design decisions and trade-offs brings disadvantages for the library users and the library engineers. While the former do not have easy access to optimized problem-specific data structures, the latter cannot extend and evolve potentially large code bases of collection libraries efficiently. Applying generative programming techniques may solve the aforementioned problems, however it requires a minimal core that is expressive enough to cover the commonalities and variability of the domain. The key enablers are data structure encodings that constitute the core of this thesis.

First, we contribute a successor to the state-of-the-art approach of Hash-Array Mapped Tries: the Compressed Hash-Array Mapped Prefix-tree (CHAMP). CHAMP improves the overall performance of immutable sets and maps by increasing cache locality and keeping the data structure canonical. Compared to its predecessor, CHAMP reduces memory footprints by design, and most notably increases runtime efficiency of iteration and equality checking significantly.

Second, we propose the Heterogeneous Hash-Array Mapped Trie (HHAMT), a generic encoding that allows storing type-heterogeneous payloads. We detail how a range of data structure design challenges can be reformulated as optimization problems for storing type-heterogeneous payloads. Amongst other optimizations, HHAMT enables highly efficient multi-maps, and collections that efficiently mix (unboxed) value-types und reference types.

Third, we discuss memory layout specialization approaches that are specific to trie-based data structures. Reducing the memory footprint of frequently used collections improves the scalability of programs that handle larger data sets, but also improves application performance in memory constraint environments.

Based on our theoretical results, we generated Capsule, a library of trie-based immutable collections that powers the Rascal programming language. Capsule's data structures carefully balance memory footprint and runtime performance, and are tailored toward JVM specifics. We managed to further reduce the performance gap between immutable and mutable collections, and to even surpass mutable collections when it comes to memory footprints and equality checking. Comparisons to state-of-the-art implementations of immutable collections in Clojure and Scala show that our encodings increase overall performance and versatility, making them sensible default choices.