CMPSC250 Final Project Proposal

Aubrey Collins and Hawk Weisman

March 13, 2016

2-3 finger trees are a data structure used to implement other common data structures in a purely functional manner.[1] Finger trees are immutable and persistent, and can be used to implement structures such as sequences, indexed sequences, and other non-tree data structures, providing improved performance over other immutable data structures such as linked lists. In this proposal, we discuss our plan to develop a finger tree implementation as a library in the Scala programming language, and to demonstrate its performance and effectiveness.

[1] Ralf Hinze and Ross Paterson. Finger trees: A simple general-purpose data structure. *J. Funct. Program.*, 16(2): 197–217, March 2006. ISSN 0956-7968. DOI: 10.1017/S0956796805005769. URL `http://dx.doi.org/10.1017/S0956796805005769`

Motivation

In functional programming, the use of *pure functions* is either encouraged or enforced. Pure functions are those which do not cause modifications to external state;[2] rather, any state changes caused by a function must be encapsulated in its return value.[3] While it may at first seem counterintuitive, requiring programmers to use only pure functions can have many benefits, such as making code more modular and easier to debug, eliminating common classes of concurrent-access errors, and enabling a wide variety of static analyses and compile-time optimizations.

In order to facilitate this style of programming, we should like to implement purely-functional data structures. Such data structures should be *immutable*, meaning that rather than being modified in-place, they are modified by creating a copy of the data structure with the changes applied.[4] As choosing to use purely-functional data structures rather than mutable ones can have performance disadvantages, we should take care to ensure that our implementations of these structures are are as efficient as possible.

[2] Often referred to as 'side effects'.

[3] Amr Sabry. What is a purely functional language? *Journal of Functional Programming*, 8:1–22, 1998. ISSN 1469-7653. URL `http://journals.cambridge.org/article_S0956796897002943`

[4] Chris Okasaki. *Purely functional data structures*. PhD thesis, Cambridge University Press, 1999

2-3 finger trees, first proposed by Hinze and Paterson, provide a way to implement sequence data structures which provide amortized $\mathcal{O}(1)$ access and $\mathcal{O}(\log n)$ split and append operations. They may be easily adapted to implement indexed or ordered sequences as well.[5] Sequence types are often implemented with linked list structures, for which accessing an arbitrary position in the sequence has a worst-case time complexity of $\mathcal{O}(n)$. Thus, we can see that the use of 2-3 finger trees is likely to offer significant performance advantages.

In the Haskell programming language, the standard library's generic sequence type (`Data.Sequence`) is implemented using finger trees.[6] Scala, a functional programming language for the Java Virtual Machine platform, has many similarities to Haskell,[7] and offers the necessary semantics to facilitate the implementation of 2-3 finger trees. While the Scala programming language offers sim-

[5] Ralf Hinze and Ross Paterson. Finger trees: A simple general-purpose data structure. *J. Funct. Program.*, 16(2): 197–217, March 2006. ISSN 0956-7968. DOI: 10.1017/S0956796805005769. URL `http://dx.doi.org/10.1017/S0956796805005769`

[6] Ross Paterson, Louis Wasserman, Bertram Felgenhauer, David Feuer, and Milan Straka. *Data.Sequence*, 2014. URL `http://hackage.haskell.org/package/containers-0.5.7.1/docs/Data-Sequence.html`. Haskell Standard Library Documentation

[7] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An overview of the Scala programming language. Technical Report IC/2004/64, École polytechnique fédérale de Lausanne, 2004

ilar generic immutable sequence types, they are implemented using either linked-lists or arrays.[8] Thus, a Scala library implementing 2-3 finger trees would be quite valuable for programmers working in that language.

[8] École polytechnique fédérale de Lausanne with contributions from Lightbend. *Scala API Documentation*, 2016. URL `http://www.scala-lang.org/api/current/index.html`

Proposal

We intend to implement and evaluate 2-3 finger trees in the Scala programming language. Our implementation will be designed in a modular manner, so that once we have a working finger tree, we can easily write multiple APIs allowing our finger tree implementation to be used as a variety of sequence-based data structures, such as a list, a queue, a stack, a deque, and an ordered sequence.

As Scala provides rich APIs for working with collections, we will ensure that the data structures we implement using our finger tree conform with the Scala sequence APIs. This will allow our finger-tree-based sequences to be used as a drop-in replacement in code that currently uses the standard library's sequence types.

Our implementation process will follow the software engineering paradigm of test-driven development. In this approach, a suite of unit tests acts as a testable specification to guide implementation. We will begin by writing a complete unit test suite that describes the expected behavior of a finger tree. Initially, since the data structure is unimplemented, all these tests will fail. As aspects of the finger tree's functionality is implemented, the corresponding unit tests will pass once the tested behavior is written correctly. Using test-driven development helps to insure that code is not left untested, as well as providing an indicator of the progress of development. It also obviates the need to write tests based on a specification, as the tests *are* the specification.

Furthermore, we will also write some short sample programs, which demonstrate the use of our data structures in real programs. These sample programs will be drawn from problems stated in the review sections in Sedgewick and Wayne's *Algorithms, Fourth Edition*.

In addition to testing for the correctness of our implementation, we will also want to assess its performance. Thus, we will write a comprehensive set of benchmarks which will test our finger-tree based collections against those from the standard library. Additionally, our sample demonstration programs could be used to measure the performance of our data structures in real applications. Since our data structures will provide the same APIs as the data structures in the Scala standard library, it will be simple to write sample code to compare finger tree-based sequences with list and array-based ones.

References

École polytechnique fédérale de Lausanne with contributions from Lightbend. *Scala API Documentation*, 2016. URL http://www.scala-lang.org/api/current/index.html.

Ralf Hinze and Ross Paterson. Finger trees: A simple general-purpose data structure. *J. Funct. Program.*, 16(2):197–217, March 2006. ISSN 0956-7968. DOI: 10.1017/S0956796805005769. URL http://dx.doi.org/10.1017/S0956796805005769.

Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An overview of the Scala programming language. Technical Report IC/2004/64, École polytechnique fédérale de Lausanne, 2004.

Chris Okasaki. *Purely functional data structures*. PhD thesis, Cambridge University Press, 1999.

Ross Paterson, Louis Wasserman, Bertram Felgenhauer, David Feuer, and Milan Straka. *Data.Sequence*, 2014. URL http://hackage.haskell.org/package/containers-0.5.7.1/docs/Data-Sequence.html. Haskell Standard Library Documentation.

Amr Sabry. What is a purely functional language? *Journal of Functional Programming*, 8:1–22, 1998. ISSN 1469-7653. URL http://journals.cambridge.org/article_S0956796897002943.