Heinrich-Heine-University Düsseldorf
Computer Science Department
Software Engineering and Programming Languages
Philipp Körner

Functional Programming – WT 2023 / 2024
Reading Guide 4: Data Structures and Laziness

Timeline: This unit should be completed by 06.11.2023.

1 Material

- Alternatives:
  - Learning Videos:
    - Implementation Details & Structural Sharing: `https://mediathek.hhu.de/watch/8a952372-1709-4`
    - Laziness: `https://mediathek.hhu.de/watch/6665ea7e-8349-4a20-a2f3-09b7e355185d`
    - ISeq vs. Seqable: `https://mediathek.hhu.de/watch/cbdd4c9e-af45-48b8-941e-f0af727a1f9`
  -
    - Rich Hickey: Clojure for Java Programmers `https://www.youtube.com/watch?v=P76Vbsk_3J0` (from 1:35:38)
    - Rich Hickey: Clojure for Java Programmers Part 2 `https://www.youtube.com/watch?v=hb3rurFxrZ8` (24:00 until 29:06)
    - Rich Hickey: Persistent Data Structures and Managed References `https://www.infoq.com/presentations/Value-Identity-State-Rich-Hickey/` (17:20 bis 32:40) (similar to the one above, more in-depth)
    - Clojure for the Brave and True, chapter 4 (Programming to Abstractions + Lazy Seqs + The Collection Abstraction)
  - The Joy of Clojure, chapter 6 (alternative to all above)
- 02_data.clj

2 Learning Outcomes

After completing this unit you should be able to

- describe the implementation of lists, vectors, sets and maps in Clojure.
- recall the runtime characteristics of various operations on lists, vectors, sets and maps.
- explain structural sharing, immutability and their interplay.
- identify possibilities for structural sharing for given data structures and operations.
- describe the concept of laziness.
- decide which calculations in Clojure are evaluated immediately and which are (or can be) delayed.
- differentiate between implicit and explicit laziness and explain the difference.

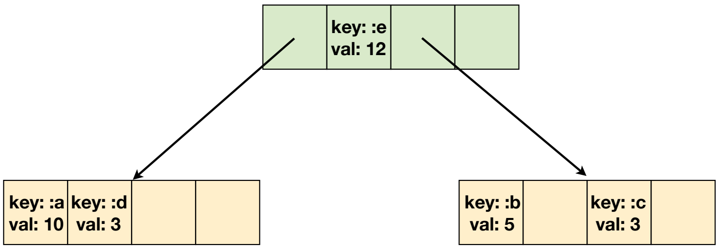
3 Highlights

- Immutability
- Structural sharing
- Laziness
- Evaluation rules, scoping
- Implementation of Hash Array-Mapped Trie (esp. path copying)

4 Exercises

Exercise 4.1 (Hash Trie)
In this exercise we consider a hash trie with a branching factor of 4, meaning every node has at most 4 children. Assume the following hash values for this exercise:

hash(:a)=	00	10	00	00	hash(:e) =	10	01	11	01
hash(:b)=	10	00	00	10	hash(:new) =	00	01	01	10
hash(:c)=	01	11	10	10	hash(:ouch)=	11	01	11	01
hash(:d)=	11	00	01	00					



a) Which map is stored in the pictured trie?

b) How many bits are needed for determining the position in an array?

c) Insert the value `:ez` under key `:new`. Which nodes can be referenced in the previous trie and which have to be copied?

d) Insert the value `:almost-a-collision` under key `:ouch`. Which nodes can be referenced in the previous trie and which have to be copied?

Exercise 4.2 (Matrix)
In the following exercise, we consider a matrix as a vector of row vectors. For example:

```
(def identity-matrix [[1 0 0] [0 1 0] [0 0 1]])
(def matrix2 [[1 0 0 1] [0 1 0 1] [0 0 1 1]])
```

a) Write a function `p!`, which outputs the matrix.

```
user=> (p! identity-matrix)
100
010
001
```

b) Write a function `trans`, which transposes the matrix, i.e. swaps the rows and columns:

```
user=> (= identity-matrix (trans identity-matrix))
true
user=> (p! (trans matrix2))
100
010
001
111
user=> (= matrix2 (trans matrix2))
false
user=> (= matrix2 (trans (trans matrix2)))
true
```

Exercise 4.3 (Black Box Testing (4clojure exercise unlock, medium #65))
Clojure has different collections, which differ (slightly) in their behaviour. Functions in `clojure.core` typically transform them into sequences and work on them. It is nonetheless important to understand the differences in behaviour and performance to choose an appropriate representation for given data.

Write a function `data-type`, which takes a collection as parameter and returns `:map`, `:set`, `:list` or `:vector` depending on which type of collection was passed.

It is prohibited to use the `list?` predicate (or similar functions). The point of this exercise is to play around with collections and understand their behaviour.

Questions

If you have any questions, please contact Philipp Körner (`p.koerner@hhu.de`) or post it to the Rocket.Chat group.