Purely Functional Data Structures
MTAT.03.271 Programming Languages Research Seminar

Kristjan Vedel

Institute of Computer Science
Faculty of Mathematics and Computer Science
University of Tartu

24.09.2012
Data Structures

A *data structure* is a particular way of organizing data, usually stored in memory and designed for better algorithm *efficiency*. The term *data structure* covers different distinct but related meanings, like:

- An *abstract data type*, that is a set of values and associated operations, specified independent of any particular implementation.
- A concrete realization or *implementation* of an abstract data type.
- An *instance* of a data type, also referred to as an *object* or *version*. 

Persistent and Ephemeral Data Structures

- **Persistent** data structures always preserve the previous versions when modified.
  - Operationally immutable, as operations do not visibly change the structure in-place. However, allows to exploit the mutable storage internally for better efficiency.
  - Can be achieved by simple copying but this is inefficient in time and space.
  - Often exploits some similarity between the new and old versions.
  - Example: Singly-linked list (using tail-sharing)

- **Ephemeral** data structures only have a single version available at a time, previous version is lost after modification.
  - Example: Dynamic array
  - In some cases asymptotically more efficient
Data Structures in Imperative and Functional languages

- **In purely functional** programming languages
  - Computation is defined as evaluation of mathematical functions
  - Destructive modifications (updates) are *not* allowed
  - *All* of the data structures in purely-functional programming languages are persistent

- **In imperative** programming languages
  - Computation is defined as statements changing the program state
  - Ephemeral data structures are usually easier to code.
  - *Most* of the data structures in imperative languages are ephemeral.

- *Purely functional data structure* is a persistent data structure with no destructive modifications allowed.
Consider two lists

\[ xs = [0, 1, 2] \]
\[ ys = [3, 4, 5] \]

Concatenation of these two lists

\[ zs = xs ++ ys \]

results in the structure where the nodes in the \( xs \) are copied, but the nodes in the \( ys \) are shared.
Purely Functional Data Structures

Usefulness:

- Referential transparency for analysis and optimization
- Better for parallel and distributed computing
- Implementing versioning, undo, rollback

Interesting topics:

- Implementation of new (efficient) purely functional data structures
- Useful techniques and purely functional data structures like finger trees and zippers for implementing other data structures.
- Considering the advantages of features like lazy evaluation
- Analyzing and verifying the time bounds or correctness of purely functional data structures