Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Рыбак Андрей Викторович

Представление структур данных индуктивными семействами и доказательства их свойств

Научный руководитель: ассистент кафедры ТП Я. М. Малаховски
Санкт-Петербург

2014

Содержание

Введение

Структуры данных используются в программировании повсеместно для упрощения хранения и обработки данных. Свойства структур данных происходят из инвариантов, которые эта структура данных соблюдает.

Практика показывает, что тривиальные структуры и их инварианты данных хорошо выражаются в форме индуктивных семейств. Мы хотим узнать насколько хорошо эта практика работает и для более сложных структур.

В данной работе рассматривается представление в форме индуктивных семейств структуры данных приоритетная очередь типа «двоичная куча».

Глава 1. Обзор предметной области

В программировании структуры данных позволяют упростить хранение и обработку множество однотипных и/или логически связанных данных. Задача структур данных — облегчить написание программ для программистов и ускорить обработку данных.

1.1. Структуры данных

Структуры данных используются в программировании для абстрагирования обработки связанных и однородных данных.

Часто используемые структуры данных включаются в стандартные библиотеки языков программирования.

1.1.1. Функциональные структуры данных

Главное отличие функциональных структур данных от императивных [1] заключается в неиспользовании разрушающих обновлений (то есть присваиваний). При обновлении структуры данных все измененные части создаются заново.

1.2. Индуктивные семейства и зависимые типы

Определение 1.1. *Индуктивное семейство* [2]— это семейство типов данных, которые могут зависеть от других типов и значений.

Тип или значение, от которого зависит зависимый тип, называют *индексом*.

Одной из областей применения индуктивных семейств являются системы интерактивного доказательства теорем.

Индуктивные семейства позволяют формализовать математические структуры, кодируя утверждения о структурах в них самих, тем самым перенося сложность из доказательств в определения.

В работах [3, 4] приведены различные подходы к построению функциональных структур данных.

Пример задания инвариантов для структуры данных — тип данных для хранения баланса в AVL-дереве [5].

Если $m \sim n$, то разница между m и n не больше чем один:

```
data _∼_ : ℕ → ℕ → Set where
  ∼+ : ∀ {n} →   n ∼ 1 + n
  ∼0 : ∀ {n} →   n ∼ n
  ∼- : ∀ {n} → 1 + n ∼ n
```

1.3. Agda

Agda [6] — чистый функциональный язык программирования с зависимыми типами. В Agda есть поддержка модулей:

```
module AgdaDescription where
```

В коде на Agda широко используются символы Unicode. Тип натуральных чисел — ℕ.

```
data ℕ : Set where
  zero : ℕ
  succ : ℕ → ℕ
```

В Agda функции можно определять как mixfix операторы. Пример — сложение натуральных чисел:

```
_+_ : ℕ → ℕ → ℕ
zero + b = b
succ a + b = succ (a + b)
```

Символы подчеркивания обозначают места для аргументов.

Зависимые типы позволяют определять типы, зависящие (индексированные) от значений других типов. Пример — список, индексированный своей длиной:

```
data Vec (A : Set) : ℕ → Set where
  nil  : Vec A zero
  cons : ∀ {n} → A → Vec A n → Vec A (succ n)
```

В фигурные скобки заключаются неявные аргументы.

Такое определение позволяет нам описать функцию head для такого списка, которая не может бросить исключение:

```
head : ∀ {A} {n} → Vec A (succ n) → A
```

У аргумента функции head тип Vec A (succ n), то есть вектор, в котором есть хотя бы один элемент. Это позволяет произвести сопоставление с образцом только по конструктору cons:

```
head (cons a as) = a
```

1.4. Выводы по главе 1

Рассмотрены некоторые существующие подходы к построению структур данных с использованием индуктивных семейств. Кратко описаны особенности языка программирования *Agda*.

Глава 2. Описание реализованной структуры данных

В данной главе описывается разработанная функциональная структура данных приоритетная очередь типа «двоичная куча».

2.1. Постановка задачи

Целью данной работы является разработка типов данных для представления структуры данных и инвариантов.

Требования к данной работе:

- Разработать типы данных для представления структуры данных
- Реализовать функции по работе со структурой данных
- Используя разработанные типы данных доказать выполнение инвариантов.

2.2. Структура данных «двоичная куча»

Определение 2.1. Двоичная куча или пирамида [7] — такое двоичное подвешенное дерево, для которого выполнены следующие три условия:

- Значение в любой вершине не больше (если куча для минимума), чем значения её потомков.
- На i-ом слое 2^i вершин, кроме последнего. Слои нумеруются с нуля.
- Последний слой заполнен слева направо (как показано на рисунке 2.1).

2.3. Тип данных для двоичной кучи

2.3.1. Заполнение дерева

Куча заполняется элементами слева. Формализуем [8] заполненность дерева:

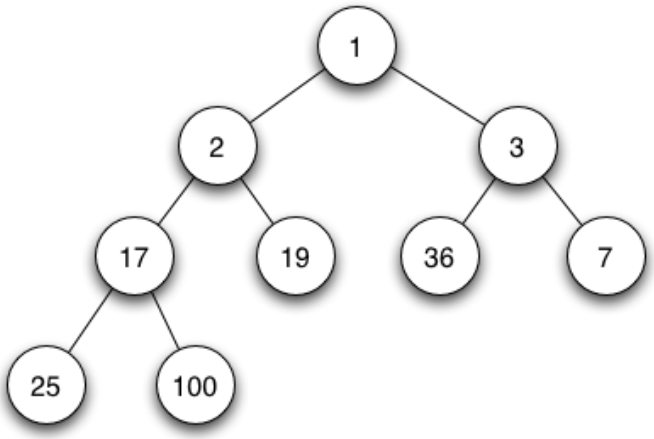

Рис. 2.1. Пример заполненной кучи для минимума

Определение 2.2. Двоичное дерево высоты h — *полное* тогда и только тогда, когда в нем $2^{h+1} - 1$ узлов.

Двоичное дерево высоты h *заполнено слева* тогда и только тогда, когда выполняется ровно один из пунктов:

- дерево — пустое
- дерево — полное (то есть оба его поддерева — либо полные высоты $h - 1$, либо пустые)
- его левое поддерево — полное высотой $h - 1$ и его правое поддерево — полное высотой $h - 2$
- его левое поддерево — заполнено слева высотой $h - 1$ и его правое поддерево — полное высотой $h - 2$
- его левое поддерево — полное высотой $h - 1$ и его правое поддерево — заполнено слева высотой $h - 1$

Определим вспомогательный тип данных для обозначения заполненности дерева:

```
data TreeState : Set where
  full almost : TreeState
```

Теперь, используя этот тип данных как дополнительный индекс и индексируя дерево его высотой, задать тип данных для заполненного слева дерева

```
data Tree : (h : ℕ) → TreeState → Set where
  et : Tree zero full - Пустое дерево
  nf : ∀ {n} → (a : Tree n full) → (b : Tree n full)
    → Tree (succ n) full - Полное дерево
  nd : ∀ {n} → (a : Tree (succ n) full) → (b : Tree n full)
    → Tree (succ (succ n)) almost - Полные поддеревья разной высоты
  nl : ∀ {n} → (a : Tree (succ n) almost) → (b : Tree n full)
    → Tree (succ (succ n)) almost - Правое поддерево - полное
  nr : ∀ {n} → (a : Tree n full) → (b : Tree n almost)
    → Tree (succ n) almost - Левое поддерево - полное
```

Для удобства будем называть заполненное слева дерево, индексированное almost *неполным*.

2.4. Вставка элементов

2.4.1. Вставка элементов в полное дерево

2.4.2. Вставка элементов в неполное дерево

2.5. Удаление элементов

2.5.1. Удаление из полного дерева

2.5.2. Удаление из неполного дерева

2.6. Выводы по главе 2

Разработаны типы данных для представления структуры данных двоичная куча.

Заключение

В данной работе реализована структура данных «двоичная куча». Разработаны типы данных для представления структуры данных и инвариантов индуктивными семействами. Реализованы функции для работы со структурой данных. Было доказано соблюдение инвариантов с использованием вспомогательных типов данных.

Таким образом, данная работа удовлетворяет поставленным требованиям.

Список литературы

1. *Okasaki C.* Purely functional data structures. Cambridge University Press, 1999. С. I–X, 1–220. ISBN: 978-0-521-66350-2.
2. *Dybjer P.* Inductive Families // Formal Asp. Comput. 1994. №4. С. 440–465.
3. *Okasaki C.* Purely Functional Data Structures. Докт. дисс. Pittsburgh, PA 15213, 1996.
4. *McBride C.* How to Keep Your Neighbours in Order. https://personal.cis.strath.ac.uk/conor.mcbride/Pivotal.p
5. *McBride C., Norell U., Danielsson N. A.* The Agda standard library — AVL trees. http://agda.github.io/agda-stdlib/html/Data.AVL.html.
6. Agda language. http://wiki.portal.chalmers.se/agda/pmwiki.php.
7. *Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.* Introduction to Algorithms, Second Edition. The MIT Press и McGraw-Hill Book Company, 2001. ISBN: 0-262-03293-7, 0-07-013151-1.
8. *Morris J.* Data Structures and Algorithms: Heaps. https://www.cs.auckland.ac.nz/~jmor159/PLDS210/heaps.