Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Mohammad Abdulaziz, Ph.D.

SS 2020
12. 7. 2019

# Functional Data Structures

Exercise Sheet 12

## Exercise 12.1 Sparse Binary Numbers

Implement operations carry, inc, and add on sparse binary numbers, analogously to the operations link, ins, and meld on binomial heaps.

Show that the operations have logarithmic worst-case complexity.

**type_synonym** *rank* = *nat*
**type_synonym** *snat* = *"rank list"*

**abbreviation** *invar* :: *"snat ⇒ bool"* **where** *"invar s ≡ sorted_wrt (<) s"*
**definition** $\alpha$ :: *"snat ⇒ nat"* **where** *"$\alpha$ s = sum_list (map ((^) 2) s)"*

**lemmas** [*simp*] = *sorted_wrt_append*

**fun** *carry* :: *"rank ⇒ snat ⇒ snat"*
**lemma** *carry_invar*[*simp*]:
  **assumes** *"invar rs"*
  **shows** *"invar (carry r rs)"*

**lemma** *carry_$\alpha$*:
  **assumes** *"invar rs"*
  **assumes** *"∀ r'∈set rs. r≤r'"*
  **shows** *"$\alpha$ (carry r rs) = 2^r + $\alpha$ rs"*
**definition** *inc* :: *"snat ⇒ snat"*
**lemma** *inc_invar*[*simp*]: *"invar rs ⟹ invar (inc rs)"*
**lemma** *inc_$\alpha$*[*simp*]: *"invar rs ⟹ $\alpha$ (inc rs) = Suc ($\alpha$ rs)"*
**fun** *add* :: *"snat ⇒ snat ⇒ snat"*
**lemma** *add_invar*[*simp*]:
  **assumes** *"invar $rs_1$"*
  **assumes** *"invar $rs_2$"*
  **shows** *"invar (add $rs_1$ $rs_2$)"*
**lemma** *add_$\alpha$*[*simp*]:
  **assumes** *"invar $rs_1$"*
  **assumes** *"invar $rs_2$"*
  **shows** *"$\alpha$ (add $rs_1$ $rs_2$) = $\alpha$ $rs_1$ + $\alpha$ $rs_2$"*
**thm** *sorted_wrt_less_sum_mono_lowerbound*

**lemma** *size_snat*:

**assumes** *"invar rs"*
**shows** *"2^length rs* $\leq$ $\alpha$ *rs + 1"*
**fun** *t_carry* :: *"rank* $\Rightarrow$ *snat* $\Rightarrow$ *nat"*
**definition** *t_inc* :: *"snat* $\Rightarrow$ *nat"*
**lemma** *t_inc_bound*:
**assumes** *"invar rs"*
**shows** *"t_inc rs* $\leq$ *log 2* ($\alpha$ *rs + 1) + 1"*
**fun** *t_add* :: *"snat* $\Rightarrow$ *snat* $\Rightarrow$ *nat"*
**lemma** *t_add_bound*:
**fixes** $rs_1$ $rs_2$
**defines** "$n_1 \equiv \alpha\ rs_1$"
**defines** "$n_2 \equiv \alpha\ rs_2$"
**assumes** *INVARS*: "*invar* $rs_1$" "*invar* $rs_2$"
**shows** "*t_add* $rs_1$ $rs_2 \leq 4 * log\ 2\ (n_1 + n_2 + 1) + 2$"

## Homework 12.1 Modified Binomial Heaps

*Submission until Friday, 17. 7. 2020, 10:00am.*

In its simplest form, a binomial heap can be implemented using binomial trees that store the the rank of every tree in its root. Such an Isabelle/HOL implementation can be found at: "src/HOL/Data_Structures/Binomial_Heaps.thy"

One optimisation is to eliminate the redundancy of storing ranks in the roots of every tree, and instead store the ranks only at the top level by pairing every tree with its rank in the heap. The following types describe a binomial heap with this optimisation.

**datatype** *'a tree = Node (root: 'a) (children: "'a tree list")*

**type_synonym** *'a heap = "(nat*'a tree) list"*

For such a heap to be a binomial heap it has to conform to the invariant *invar* defined as follows:

**fun** *invar_btree* :: *"nat* $\Rightarrow$ *'a::linorder tree* $\Rightarrow$ *bool"* **where**
*"invar_btree r (Node x ts)* $\longleftrightarrow$
*length ts = r* $\wedge$ ($\forall$ *(r',t)*$\in$*set (zip (rev [0..<r]) ts). invar_btree r' t)"*

**definition** *invar_bheap* :: *"'a::linorder heap* $\Rightarrow$ *bool"* **where**
*"invar_bheap ts*
$\longleftrightarrow$ ($\forall$ *(r,t)*$\in$*set ts. invar_btree r t)* $\wedge$ *(sorted_wrt (<) (map fst ts))"*

**fun** *invar_otree* :: *"'a::linorder tree* $\Rightarrow$ *bool"* **where**
*"invar_otree (Node x ts)* $\longleftrightarrow$ ($\forall$ *t*$\in$*set ts. invar_otree t* $\wedge$ *x* $\leq$ *root t)"*

**definition** *invar_oheap* :: "'a::linorder tree list ⇒ bool" **where**
"invar_oheap ts ⟷ (∀ t∈set ts. invar_otree t)"

**definition** *invar* :: "'a::linorder heap ⇒ bool" **where**
"invar ts ⟷ invar_bheap ts ∧ invar_oheap (map snd ts)"

In this homework you are required to define an insertion and a merging functions for this heap and show that they preserve the elements of their inputs as well as produce heaps that conform to the invariant *invar*.

**definition** *insert* :: "'a::linorder ⇒ 'a heap ⇒ 'a heap" **where**

**lemma** *invar_insert*[*simp*]: "invar t ⟹ invar (insert x t)"

**lemma** *mset_heap_insert*[*simp*]: "mset_heap (insert x t) = {#x#} + mset_heap t"

**fun** *merge* :: "'a::linorder heap ⇒ 'a heap ⇒ 'a heap" **where**

**lemma** *invar_merge*[*simp*]: "⟦ invar $ts_1$; invar $ts_2$ ⟧ ⟹ invar (merge $ts_1$ $ts_2$)"

**lemma** *mset_heap_merge*[*simp*]: "mset_heap (merge $ts_1$ $ts_2$) = mset_heap $ts_1$ + mset_heap $ts_2$"

Hint: you can start with the theory file "src/HOL/Data_Structures/Binomial_Heaps.thy" that has an implementation of binomial heaps without this optimisation, and then edit it.

## **Homework 12.2** Be Original!

*Submission until Friday, 17. 7. 2020, 10:00am.*

Develop a nice Isabelle formalisation yourself!

- This homework goes in parallel to other homeworks for the rest of the lecture period. From next sheet on, we will reduce regular homework load a bit, such that you have a time-frame of 3 weeks with reduced regular homework load.
- This homework will yield 15 points (for minimal solutions). Additionally, up to 15 bonus points may be awarded for particularly nice/original/etc solutions.
- You may develop a formalisation from all areas, not only functional data structures.
- Document your solution, such that it is clear what you have formalised and what your main theorems state!
- Set yourself a time frame and some intermediate/minimal goals. Your formalisation needs not be universal and complete after 3 weeks.
- You are welcome to discuss the realisability of your project with the tutor or ask him for possible ideas!

- Should you need inspiration to find a project: Sparse matrices, skew binary numbers, arbitrary precision arithmetic (on lists of bits), interval data structures (e.g. interval lists), spatial data structures (quad-trees, oct-trees), Fibonacci heaps, prefix tries/arrays and BWT, etc.