Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Fabian Huch

SS 2021
18. 6. 2021

Functional Data Structures

Exercise Sheet 10

Exercise 10.1 Union Function on Tries

Define a function to merge two tries and show its correctness

fun *union* :: *"trie ⇒ trie ⇒ trie"*
lemma *"isin (union a b) x = isin a x ∨ isin b x"*

Exercise 10.2 Tries with 2-3-trees

In this exercise, you shall develop a trie data structure for keys of type *'a list* (instead of *bool list*).

Thus, a node needs to store a map from *'a* to the next trie.

In a first step, we encode the map as *'a ⇒ 'b option*

datatype *'a trie = Leaf | Node bool "'a ⇀ 'a trie"*

Define and prove correct membership, insertion and deletion (without shrinking the trie).

fun *isin* :: *"'a trie ⇒ 'a list ⇒ bool"*
fun *ins* :: *"'a list ⇒ 'a trie ⇒ 'a trie"*
lemma *ins_correct*: *"isin (ins as t) bs = (as=bs ∨ isin t bs)"*

fun *delete* :: *"'a list ⇒ 'a trie ⇒ 'a trie"*
lemma *delete_correct*: *"isin (delete as t) bs = (as ≠ bs ∧ isin t bs)"*

Now refine the trie data structure to use 2-3-trees for the map. Note: To make the provided interface more usable, we introduce some abbreviations here:

abbreviation *"empty23 ≡ Tree23.Leaf"*
abbreviation *"inv23 t ≡ complete t ∧ sorted1 (inorder t)"*

The refined trie datatype

datatype *'a trie' = Leaf' | Node' bool "('a×'a trie') tree23"*

Define an invariant for trie' and an abstraction function to trie. Then define membership, insertion, and deletion, and show that they behave correctly wrt. the abstract trie. Finally, combine the correctness lemmas to get a set interface based on 2-3-tree tries.

You will need a lemma like the following for termination:

lemma *lookup_size_aux*[*termination_simp*]:
"lookup m k = Some v ⟹ size (v::'a trie') < Suc (size_tree23 (λx. Suc (size (snd x))) m)"

fun *trie'_inv* :: *"'a::linorder trie' ⇒ bool"*
fun *trie'_α* :: *"'a::linorder trie' ⇒ 'a trie"*
fun *isin'* :: *"'a::linorder trie' ⇒ 'a list ⇒ bool"*
fun *ins'* :: *"'a::linorder list ⇒ 'a trie' ⇒ 'a trie'"*

fun *delete'* :: *"'a::linorder list ⇒ 'a trie' ⇒ 'a trie'"*

lemmas *map23_thms*[*simp*] = *M.map_empty Tree23_Map.M.map_update Tree23_Map.M.map_delete Tree23_Map.M.invar_empty Tree23_Map.M.invar_update Tree23_Map.M.invar_delete M.invar_def M.inorder_update M.inorder_inv_update sorted_upd_list*

lemma *ins'_correct*: *"trie'_inv t ⟹ (isin' (ins' xs t) ks ⟷ xs=ks ∨ isin' t ks) ∧ trie'_inv (ins' xs t)"*

lemma *delete'_correct*: *"trie'_inv t ⟹ (isin' (delete' xs t) ks ⟷ xs≠ks ∧ isin' t ks) ∧ trie'_inv (delete' xs t)"*

Homework 10.1 Intermediate Tries as Arrays

Submission until Thursday, June 24, 23:59pm.

The following trie definition is in between of normal tries and Patricia tries:

datatype *'a trieIP = LfIP*
| *UnIP bool "'a trieIP"*
| *NdIP "'a option" "'a trieIP ∗ 'a trieIP"*

Moreover, it stores associated elements (as *'a::type option*s) instead of membership only. As such, we can use this definition to represent array-like operations.

Take the abstraction function between those intermediate tries and normal tries:

fun *keys* :: *"'a trieIP ⇒ trie"*

Define lookup, update, and delete (without shrinking), and prove them correct:

fun *lookupIP* :: *"'a trieIP ⇒ bool list ⇀ 'a"*
theorem *lookup_keys*: *"isin (keys t) ks ⟷ is_some (lookupIP t ks)"*

fun *updateIP* :: *"bool list ⇒ 'a ⇒ 'a trieIP ⇒ 'a trieIP"*
theorem *update_set*: *"x ∈ set_trieIP (updateIP ks x t)"*
theorem *update_lookup*: *"lookupIP (updateIP ks x t) ks = Some x"*

fun *delete0IP* :: *"bool list ⇒ 'a trieIP ⇒ 'a trieIP"*
lemma *delete0_keys*: *"keys (delete0IP ks t) = delete0 ks (keys t)"*

Homework 10.2 (Bonus, 2 pts) Deletion with shrinking

Submission until Thursday, June 24, 23:59pm. Define a delete function which removes an elements from *trieIP* and which shrinks a node when necessary.

Hint: you will need two shrink functions for the two different kinds of non-leaf nodes in *trieIP*.

fun *deleteIP* :: *"bool list ⇒ 'a trieIP ⇒ 'a trieIP"*

Show that this function is equivalent to the function *delete*, which is a delete function with shrinking for binary tries.

lemma *delete_keys*: *"keys (deleteIP ks t) = delete ks (keys t)"*

Homework 10.3 Be Original!

Submission until Thursday, July 8, 23:59pm.

Develop a nice Isabelle formalisation yourself!

- This homework goes in parallel to other homeworks for most of the remaining lecture period. We will reduce regular homework load (the sheets are half the size/points), such that you have a time-frame of 3 weeks with reduced regular homework load. You should choose your topic until next week, and have an idea of what to formalize.
- The 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 encouraged to discuss the realisability of your project with us!
- 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. You can also ask the tutor for possible ideas.