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

SS 23
27. 6. 2023

Functional Data Structures

Exercise Sheet 10

Exercise 10.1 Union Function on Tries

Define a function to union 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 the lecture, tries stored child nodes with an abstract map. We want to 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 ≡ Leaf"*
abbreviation *"inv23 t ≡ complete t ∧ sorted1 (inorder t)"*

The refined trie datatype:

datatype *'a trie' = Nd' bool "('a×'a trie') tree23"*

Define an invariant for trie' and an abstraction function to trie. Based on the original tries, 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"*
definition *empty'* :: *"'a trie'"* **where**
[*simp*]: *"empty' = Nd' False empty23"*

fun *isin'* :: *"'a::linorder trie' ⇒ 'a list ⇒ bool"*
fun *insert'* :: *"'a::linorder list ⇒ 'a trie' ⇒ 'a trie'"*
fun *delete'* :: *"'a::linorder list ⇒ 'a trie' ⇒ 'a trie'"*

definition *set′* :: "′*a*::*linorder trie′* ⇒ ′*a list set*" **where**
[*simp*]: "*set′ t* = *set* (*trie′_α t*)"

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*

interpretation *S′*: *Set*
where *empty* = *empty′* **and** *isin* = *isin′* **and** *insert* = *insert′* **and** *delete* = *delete′*
and *set* = *set′* **and** *invar* = *trie′_inv*
proof (*standard*, *goal_cases*)

Homework 10.1 Tries with Same-Length Keys (8 points)

Submission until Monday, July 3, 23:59pm.

Consider the following trie datatype:

datatype *trie′* = *LfF* | *LfT* | *NdI* (*trie′* × *trie′*)

It is meant to store keys of the same length only. Thus, the *NdI* constructor stores inner nodes, and there are two types of leaves, *LfF* if this path is not in the set, and *LfT* if it is in the set.

Define an invariant *is_trie N t* that states that all keys in *t* have length *N*, and that there are no superfluous nodes, i.e., no nodes of the form *NdI* (*LfF*, *LfF*).

fun *is_trie* :: "*nat* ⇒ *trie′* ⇒ *bool*"

Hint: The following should evaluate to true!

value "*is_trie 42 LfF*"
value "*is_trie 2* (*NdI* (*LfF*,*NdI* (*LfT*,*LfF*)))"

Whereas these should be false

value "*is_trie 42 LfT*"
value "*is_trie 2* (*NdI* (*LfT*,*NdI* (*LfT*,*LfF*)))"
value "*is_trie 1* (*NdI* (*LfT*,*NdI* (*LfF*,*LfF*)))"

Define membership, insert, and delete functions, and prove them correct!

fun *isin* :: "*trie′* ⇒ *bool list* ⇒ *bool*"
fun *ins* :: "*bool list* ⇒ *trie′* ⇒ *trie′*"
lemma *isin_ins*:
  assumes "*is_trie n t*"
   and "*length as* = *n*"
 shows "*isin* (*ins as t*) *bs* = (*as* = *bs* ∨ *isin t bs*) ∧ *is_trie n* (*ins as t*)"

fun *delete* :: "*bool list* ⇒ *trie′* ⇒ *trie′*"
lemma *isin_delete*:
  assumes "*is_trie n t*"
 shows "*isin* (*delete as t*) *bs* = (*as*≠*bs* ∧ *isin t bs*) ∧ (*is_trie n* (*delete as t*))"

Hints:

- Like in the *delete* function for standard trie's, you may want to define a "smart-constructor" $node :: trie' \times trie' \Rightarrow trie'$ for nodes, that constructs a node and handles the case that both successors are *LfF*.
- Consider proving auxiliary lemmas about the smart-constructor, instead of always unfolding it with the simplifier.

Homework 10.2 Be Original!

Submission until Monday, July 10, 23:59pm.

Develop a nice Isabelle formalisation yourself!

- You may develop a formalisation from all areas, not only functional data structures. Creative topics are encouraged!
- Document your solution well, 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.
- You are encouraged to discuss the realisability of your project with us!
- Pick a topic this week (the regular homework is a bit shorter). Next week, the project will be the exclusive task.
- In total, the homework will yield 15 points (for minimal solutions). Additionally, bonus points may be awarded for particularly nice/original/etc solutions.