Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

SS 2017
23. 6. 2017

Functional Data Structures

Exercise Sheet 9

Exercise 9.1 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 9.2 Intermediate Abstraction Level for Patricia Tries

We introduce an abstraction level in between tries and Patricia tries: A node with only a single non-leaf successor is represented as an unary node.

Via unary nodes, this implementation already introduces a notion of common prefix, but does not yet summarize runs of multiple prefixes into a list.

datatype *itrie = LeafI | UnaryI bool itrie | BinaryI bool "itrie * itrie"*

fun *abs_itrie* :: *"itrie ⇒ trie"* — Abstraction to tries
where
"abs_itrie LeafI = Leaf"
| *"abs_itrie (UnaryI True t) = Node False (Leaf, abs_itrie t)"*
| *"abs_itrie (UnaryI False t) = Node False (abs_itrie t, Leaf)"*
| *"abs_itrie (BinaryI v (l,r)) = Node v (abs_itrie l, abs_itrie r)"*

Refine the union function to intermediate tries

fun *unionI* :: *"itrie ⇒ itrie ⇒ itrie"*

Next, we define an abstraction function from Patricia tries to intermediate tries. Note that we need to install a custom measure function to get the termination proof through!

fun *absI_ptrie* :: *"ptrie ⇒ itrie"* **where**
"absI_ptrie LeafP = LeafI"
| *"absI_ptrie (NodeP [] v (l,r)) = BinaryI v (absI_ptrie l, absI_ptrie r)"*
| *"absI_ptrie (NodeP (x#xs) v (l,r)) = UnaryI x (absI_ptrie (NodeP xs v (l,r)))"*

Warmup: Show that abstracting Patricia tries over intermediate tries to tries is the same as abstracting directly to tries.

lemma *"abs_itrie o absI_ptrie = abs_ptrie"*

Refine the union function to Patricia tries.

Hint: First figure out how a Patricia trie that correspond to a leaf/unary/binary node looks like. Then translate *unionI* equation by equation!

More precisely, try to find terms *unaryP* and *binaryP* such that

$$absI\_ptrie\ (unaryP\ k\ t) = UnaryI\ k\ (absI\_ptrie\ t)$$

$$absI\_ptrie\ (binaryP\ v\ (l,\ r)) = BinaryI\ v\ (absI\_ptrie\ l,\ absI\_ptrie\ r)$$

You will encounter a small problem with *unaryP*. Which one?

fun *unionP* :: *"ptrie ⇒ ptrie ⇒ ptrie"*
lemma *"absI_ptrie (unionP* t_1 t_2*) = unionI (absI_ptrie* t_1*) (absI_ptrie* t_2*)"*

Homework 9.1 Shrunk Trees

Submission until Friday, 30. 6. 2017, 11:59am.

Have a look at the *delete2* function for tries. It maintains a "shrunk" - property on tries. Identify this property, define a predicate for it, and show that it is indeed maintained by empty, insert, and delete2!

fun *shrunk* :: *"trie ⇒ bool"*
lemma *"shrunk Leaf"*
lemma *"shrunk t ⟹ shrunk (insert ks t)"*
lemma *"shrunk t ⟹ shrunk (delete2 ks t)"*

Homework 9.2 Refining Delete

Submission until Friday, 30. 6. 2017, 11:59am.

Refine the delete function to intermediate tries and further down to Patricia tries.

fun *deleteI* :: *"bool list ⇒ itrie ⇒ itrie"* **where**
lemma *"abs_itrie (deleteI ks t) = delete ks (abs_itrie t)"*
fun *pdelete* :: *"bool list ⇒ ptrie ⇒ ptrie"*
lemma *"absI_ptrie (pdelete ks t) = deleteI ks (absI_ptrie t)"*