Exercise 8.1 Abstract Set Interface

In Isabelle/HOL we can use a so called locale to model the abstract set interface. The locale fixes the operations as parameters, and makes assumptions on them.

locale set_interface = fixes invar :: "'s ⇒ bool" and α :: "'a set" fixes empty :: 's assumes empty_invar: "invar empty" and empty_α: "α empty = {}" fixes is_in :: "'a ⇒ 's ⇒ bool" assumes is_in_invar: "invar s =⇒ is_in s x ⇔ x ∈ α s" fixes ins :: "'a ⇒ 's ⇒ 's" assumes ins_invar: "invar s =⇒ invar (ins x s)" and ins_α: "invar s =⇒ α (ins x s) = Set.insert x (α s)" fixes to_list :: "'s ⇒ 'a list" assumes to_list_invar: "invar s =⇒ set (to_list s) = α s" begin

Inside the locale, all the assumptions are available

thm empty_invar empty_α is_in_α ins_invar ins_α to_list_α

Note that you know nothing about the structure of the fixed parameters or the types 'a and 's!

We can define a union function as follows:

definition union :: "'s ⇒ 's ⇒ 's" where "union A B = fold ins (to_list A) B"

Show the interface specification for union:

lemma union_invar:
  assumes "invar A"
  assumes "invar B"
  shows "invar (union A B)"
lemma union_α:
  assumes "invar A"
  assumes "invar B"
  shows "α (union A B) = α A ∪ α B"

Define an intersection function and show its interface specification

definition intersect :: "'s ⇒ 's ⇒ 's"
lemma intersect_invar:
lemma intersect_inv:
end

Having defined the locale, we can instantiate it for implementations of the set interface.
For example for BSTs:
interpretation bst_set: set_interface bst set_tree Tree.Leaf "BST_Demo.isin" "BST_Demo.ins"
  Tree.inorder
  apply unfoldlocales

Show the goals

Now we also have instantiated versions of union and intersection

term bst_set.union
thm bst_set.union_α bst_set.union_invar

term bst_set.intersect
thm bst_set.intersect_α bst_set.intersect_invar

Instantiate the set interface also for:
- Distinct lists
- 2-3-Trees

**Homework 8.1** Estimating the Size of 2-3-Trees

*Submission until Friday, 23. 6. 2017, 11:59am.*

Show that for 2-3-trees, we have:

\[
\log_3 (s(t) + 1) \leq h(t) \leq \log_2 (s(t) + 1)
\]

Hint: It helps to first raise the two sides of the inequation to the 2nd/3rd power. Use sledgehammer and find-theorems to search for the appropriate lemmas.

lemma height_est_upper: "bal t ⇒ height t ≤ log 2 (size t + 1)"
lemma height_est_lower: "bal t ⇒ log 3 (size t + 1) ≤ height t"

Define a function to count the number of leaves of a 2-3-tree
Define a function to determine whether a tree only consists of 2-nodes and leaves:

\[
\text{fun is}_2\text{Tree :: "tree23 ⇒ bool"}
\]

Show that a 2-3-tree has only 2 nodes, if and only if its number of leafs is 2 to the power of its height!

\[
\text{Hint: The } \rightarrow \text{ direction is quite easy, the } \leftarrow \text{ direction requires a bit more work!}
\]

\[
\text{lemma } \text{bal } t \implies \text{is}_2\text{Tree } t \iff \text{numLeaves } t = 2^{\text{height } t}
\]

**Homework 8.2 Deforestation**

*Submission until Friday, 23. 6. 2017, 11:59am.*

A disadvantage of using the generic algorithms from the set interface for binary trees (and other data structures) is that they construct an intermediate list containing the elements of one tree.

Define a function that folds over the in-order traversal of a binary tree directly, without constructing an intermediate list, and show that it is correct.

\[
\text{Note: Optimizations like this are called deforestation, as they get rid of intermediate tree-structured data (in our case: lists which are degenerated trees).}
\]

\[
\text{fun foldTree :: "('a ⇒ 's ⇒ 's) ⇒ 'a tree ⇒ 's"}
\]

\[
\text{lemma } \text{foldTree } f t s = \text{fold } f \text{ (Tree.inorder } t \text{) } s
\]

**Homework 8.3 Bit-Vectors**

*Submission until Friday, 23. 6. 2017, 11:59am. Bonus Homework (3p)*

A bit-vector is a list of Booleans that encodes a finite set of natural numbers as follows: A number \(i\) is in the set, if \(i\) is less than the length of the list and the \(i\)th element of the list is true.

For 3 bonus points, define the operations empty, member, insert, and to-list on bit-vectors, and instantiate the set-interface from Exercise 1!

\[
\text{type synonym bv = "bool list"}
\]

\[
\text{definition bvα :: "bv ⇒ nat set"}
\]

\[
\text{where } \text{bvα } l = \{ \text{ i. } i < \text{length } l \land l[i] \}
\]

\[
\text{definition bv_empty :: bv}
\]

\[
\text{definition bv_member :: "bv ⇒ nat ⇒ bool"}
\]

\[
\text{definition bv_ins :: "nat ⇒ bv ⇒ bv"}
\]

\[
\text{definition bv_to_list :: "bv ⇒ nat list"}
\]

\[
\text{interpretation bv_set: set_interface "λ_ True" bvα bv_empty bv_member bv_ins bv_to_list}
\]