Homework 4
Phantom Types

98-317: Hype for Types

Due: 2 Oct 2018 at 6:30 PM

1 Introduction

In class, we discussed the idea that you can add useless type parameters to a type to increase the number of correctness properties you can statically check. In this homework, you will take a signature for an Array structure, and add phantom types to it to statically enforce the when the arrays can be modified.

Turning in the Homework You should submit a PDF with your solutions to Autolab under Phantom Types.

2 Spooky Arrays

Consider the following signature for a structure representing Arrays in SML.

```
signature ARRAY = sig
  type 'a array

  (* get A i returns SOME (A[i]) if i is in bounds, and NONE otherwise *)
  val get : 'a array -> int -> 'a option

  (* set A i x sets the ith index of the array to x *)
  val set : 'a array -> int -> 'a -> unit

  (* For each A[i], map_inplace f A sets A[i] to f(A[i]) *)
  val map_inplace : ('a -> 'a) -> 'a array -> unit

  (* map f A creates a new array B in which B[i] is f(A[i]) *)
  val map : ('a -> 'b) -> 'a array -> 'a array

  (* create i x creates a new array of length i where every element is x *)
  val create : int -> 'a -> 'a array

  (* Makes the array readonly *)
  val to_readonly : 'a array -> 'a array
end
```

Arrays should behave like they do in other languages like C and Python: you can find an element at a particular index in O(1) and you can modify an element at any index in O(1). In SML, we also have higher-order functions, so we want to be able to `map` on arrays, and also `map_inplace`, which is like map but instead of making a new array, changes the elements of the array in-place.

Where do phantom types come in here? We'd like to be able to designate an array as read-only or not: read-only arrays should not be able to be modified, so set and map_inplace should not work on them. This is useful if we want some untrustworthy functions to only be able to look at the contents of our array (and thus get the nice O(1) lookup properties of arrays), not modify them.

Your job is to make a new signature, `PHANTOM_ARRAY`, which enforces the difference between read-only and read-write arrays. We've gotten you started off by providing the type declarations. It's your job to fill in the types of values in the signature.

```
signature PHANTOM_ARRAY = sig
   type readwrite
   type readonly
   type ('a, 'b) array

   val get :           (* Problem 1 *)
   val set :           (* Problem 2 *)
   val map_inplace :   (* Problem 3 *)
   val map :           (* Problem 4 *)
   val create :        (* Problem 5 *)
   val to_readonly :   (* Problem 6 *)
end
```

For each problem, write down the type that the value should have. The types should enforce the following properties:

1. If `A` is an array, then `set (to_readonly A) i x` should not typecheck.
2. If `A` is an array, then `map_inplace f (to_readonly A)` should not typecheck.
3. If `A` is an array and `i` is an int, then `get A i` should typecheck.
4. If `A` is an array containing values of type `t` and `f` : `t` $\rightarrow$ `t`, then `map f A` should typecheck and should return an array that is readonly if `A` is readonly.
5. If `A` is an array that is not read-only containing values of type `t` and `f` : `t` $\rightarrow$ `t`, then `map_inplace f A` should typecheck.
6. If `i` is an int and `x` is a value of type `t`, then `create i x` should typecheck and should return a read-write array that contains values of type t.
7. If `A` is an array, then `to_readonly A` should typecheck and should return a read-only array.