Proposal for student research project: Vector Implementations in Coq, a comparison

Introduction

Collection data types with fixed length that can be indexed are a very common thing in most programming languages. One Problem with those data types often is tha lack of protections against the use of indices outside the bounds of the collection. Because this can lead to various problems in critical applications one may want to use a proof assistant. To do so the data structure has to be modeled inside the given proof assistant of choice, but there are different ways to do so. This research project aims to compare different implementations for the Vector type in coq which is a dependently typed collection with the length as parameter of the type fixing it to a given length.

Considered data structures

Inductively defined Vectors

Vectors are an inductive Datatype with the constructors nil for empty vectors and cons to construct longer vectors.

```
Inductive vector (A : Type) : nat -> Type :=
  | nil : vector A 0
  | cons : forall (h : A) (n : nat), vector A n -> vector A (S n).
```

this is the implementation used in the Coq standard library.

Functional defined Vectors

A Vector is a function that takes a bounded numeral and returns the corresponding element for this numeral.

```
Definition vector (A : Type) (n : nat) := Fin.t n -> A.
```

As List

The Vector type is a Record containing a list and a proof that the length of this list is as specified. Implementation:

```
Record vector (A : Type) (n : nat) :=
  mk_vector { elts : list A; elts_spec : length elts = n }.
```

Recursive Tuples

The Vector type is represented by nested tuples. The empty vector is simply the empty tuple the vector with one element a unary tuple and for further length more tuples are nested inside of each other.

```
Fixpoint vector (A : Type) (n : nat) : Type :=
  match n with
  | 0 => unit
  | S n => A * (vector A n)
  end.
```

Functions to implement

The relevant functions to be implemented are the following:

- nil
- cons
- vhd
- tl
- last
- const
- nth
- replace
- take
- append
- rev
- map
- fold_right
- of_list
- to_list

They will be implemented with the same type signature as the corresponding functions in the standard library just with changed vector types. Also the vector_ind Lemma will be shown for all implementations.

Lemmas

Beside the implementations of the data types also their properties in proofing lemmas for them will be compared. For this some Lemmas included in the standard library module VectorSpec will be proofed for the implementations.

Comparison

On basis of the implementations and the lemmas the different data types will be compared in which situation it is easier to use one or the other. Because this is no strictly defined metric we will discuss different aspects like amount of code needed, know how needed, etc.. It is important to say, that it still is a some what subjective comparison.