

Chapter 5

Program extraction

Constructive functional programming is done in three steps, the last one being program extraction:

1. program specification
2. proof of existence
3. program extraction

In the practical work we consider the specification of a sorting algorithm from which we extract a program for insertion sort.

5.1 Program specification

Here we discuss the general form of a specification. If we are concerned with program abstraction we often consider formulas of the following form:

$$\forall a : A. P(a) \rightarrow \exists b : B. Q(a, b)$$

with P a predicate on A and Q a predicate on $A \times B$, and consider such formula as a specification. According to the interpretation discussed above, a program that verifies this specification should do the following:

It maps an input $a : A$ to an object $b : B$ together with a program that transforms a program that verifies $P(a)$ into a program that verifies $Q(a, b)$.

The aim of program extraction is to obtain from a proof of a formula of the form $\forall a : A. P(a) \rightarrow \exists b : B. Q(a, b)$ a function or program

$$f : A \rightarrow B$$

with the following property:

$$\forall a : A. P(a) \rightarrow Q(a, f(a)).$$

This is not possible for all formulas. Coq contains a module, called `Extraction`, that permits one to extract in some cases a program from a proof.

5.2 Proof of existence

Most of the work is in this part. The style and contents of the program that is extracted in the third step originate from the style and contents of the proof of existence in the second step.

5.3 Program extraction

This part is automated. Roughly, the part of the proof which is about proving is erased. This is the part in `Prop`. The part of the proof which is about calculating remains. This is the part in `Set`. The extracted program is CAML or Haskell. Recently a new extraction module was written for Coq, for more information see the homepage of the Coq project.

5.4 Insertion sort

We consider a formula that can be seen as the specification of a sorting algorithm. We assume the type `natlist` for the finite lists of natural numbers, and in addition the following predicates:

- The predicate `Sorted` on `natlist` that expresses that a list is sorted (that is, the elements form an increasing list of natural numbers).
- The predicate `Permutation` taking two elements of `natlist` as input, that expresses that those two lists are permutations of each other (that is, they contain exactly the same elements but possibly in a different order).
- The predicate `Inserted` is used in the definition of `Permutation`. It takes as input a natural number, a `natlist`, and another `natlist`. It expresses that the second `natlist` is obtained by inserting the natural number in some position in the first `natlist`.

The formula or specification we are interested in is then the following:

Theorem `Sort` : forall l : natlist,
 {l' : natlist | Permutation l l' /\ Sorted l'}.

Note the use of the existential quantification in `Set` (not in `Prop`): this is essential for the use of `Extraction`. The aim of the practical work of this week is to prove this formula correct, and extract from the proof a mapping

$$f : \text{natlist} \rightarrow \text{natlist}$$

in the form of a ML program, that verifies the following:

$$\forall l : \text{natlist}. \text{Sorted}(f(l)) \wedge \text{Permutation}(l, f(l)).$$

Proof of Sort. The proof of sort proceeds by induction on the natlist l . The base case (for the empty list), follows from the definition of Sorted and the definition of Permutation. In the induction step, the predicates Insert and Permutation are used, in a somewhat more difficult way.

5.5 Coq

Existential quantification in Coq. There are two different ways to write an existential quantification $\exists x : A. P$ in Coq:

`exists x:A, P`

with `exists x:A, P : Prop`, and

`{x:A | P}`

with `{x:A | P} : Set`. The existential quantification `exists x:A, P` in `Prop` is considered as a logical proposition without computational information. It is used only in logical reasoning. The existential quantification `{x:A | P}` in `Set` is considered as the type of the terms in `A` that verify the property `P`. It is used to extract a program from a proof.