Institut Polytechnique de Paris
Master 2 - Master Parisien de Recherche en Informatique

February 24, 2023
2.7.2 - Proof assistants

Polynomials and Boolean Formulas

TABET GONZALEZ Salwa

Previous experience with proof assistants

I hadn't used a proof assistant previous to this course, other than a few hours on Agda. Hence, my proofs are surely inelegant, convoluted and very long.

1 Multivariate Polynomials

1.1 Valid Polynomials

a) The first attempt at solving this question used three pattern matchings. However, when trying to prove the equivalence between the propositional validity checking predicate and the boolean one, unfolding the various patterns was tedious.

```
Fixpoint is_valid_i (p':poly) (j:nat) : Prop :=
match p' with
| Cst z => True
| Poly p i q =>
  match Nat.ltb j i with
  | false => False
  | true =>
    match q with
    |Cst 0 => False
    |_ => is_valid_i p (S i) /\ is_valid_i q i
end end end.

Definition is_valid (p:poly) : Prop :=
  is_valid_i p 0.
```

Then, I separated some validity checks (such as verifying $q \neq 0$) and reduced the pattern-matching to a simple one. The new auxiliary function is an inductive predicate and accounts for the indices j and i, but `is_valid` isn't.

b) Similarly to the previous question, my first attempt at writing the predicate involved three pattern-matchings, but was considerably simplified for proving purposes. Then for proving that this boolean predicate is equivalent to the inductive one, I prove first the equivalence for the predicates that take into account i, and use this lemma for proving equivalence in the case where $i = 0$.

c) In order to prove that two valid polynomials with the same structure are equal with respect to Leibniz equality, I must first prove the proof irrelevance lemma. I adapted a proof from the module `Eqdep_dec`, and simplified it to treat only the proofs of `true = true`.

1.2 Coefficients and Values

a) I use a similar structure as valid polynomials to encode monomials. Therefore, there is a record `valid_mono` that takes a monomial and a proof that such monomial is valid. A monomial is valid if it is equal to 1 or it is of the form $X_i \cdot m$ with m a valid monomial uch that every X_j appearing in m satisfies $j \geq i$.

Then, the predicate `get_coeff` follows the same approach as the previous predicates : there is a base function that does all the work (here, for a polynomial p and a monomial m) and the predicate uses such function for valid arguments.

b) To prove that two valid polynomials with the same coefficients are equal, there is a certain amount of intermediate lemmas I had to prove beforehand.

The first lemma `exists_coeff` proves that a nonconstant polynomial has a nonzero coefficient. As often, I proved this by first proving an auxiliary lemma `exists_coeff_i`, which takes into account the predicate `valid_bool_i`, and can be proven by induction on the polynomial. I then proved the lemma `coeff_poly`, which allows us to prove the general case of the induction, and finally, I proved the theorem `eq_coeff`, which states that if two polynomials p and q have the same coefficients, then they are equal, by induction on p and q.

c) I defined the evaluation of a polynomial at a point represented by a function `f : nat -> Z`, where `f i` represents the value we attribute to X_i. This function is defined by induction on the polynomial in a very intuitive way.

d) I then proved the theorem `eval_eq`, which says that two valid polynomials which take the same values everywhere are equal. Since I did not manage to prove it directly by induction on the polynomials, I first proved the stronger property that, if two valid polynomials in the variables $(X_j)_{j \geq i}$, then there exists a function `f : nat -> Z` such that, for large enough n, the evaluations of the two polynomials at the function

$$j \mapsto \begin{cases} n & \text{if } i = j \\ f(j) & \text{otherwise} \end{cases}$$

are different. I first proved the lemma `non_constant_poly`, which is this property in the case when one of the two polynomials is constant, and then the lemma `diff_poly`, which is the general case.

1.3 Arithmetic Operations

a) I defined the sum of two polynomials by induction on p and q. The first difficulty was finding a way to show Coq that the induction is valid, i.e. that the function is called recursively on strictly smaller cases. To do so, I defined inductively the height of a polynomial, and then defined the sum by taking the sum of the heights of p and q into account. Then, I had to ensure that the result was still valid, which I did by defining the function `validify`, which takes a polynomial and recursively removes zero nodes. After that, it was intuitive to show that the sum is compatible with evaluation.

b) I wasn't able to complete this question.

2 Boolean Tautologies

2.1 Boolean Formulas

For these questions, I just followed the steps shown in the Ltac example.

2.2 Polynomial Reflection

a) Because I didn't implement the product of two polynomials, I couldn't do the following questions. However, I was able to define the negation of a polynomial, with the predicate `minus`.