Master Parisien de Recherche en Informatique 2022–2023
Proof Assistants

Project – Polynomials and Boolean Formulas

The goal of this project is to write an automated tactic for proving boolean tautologies by reflecting them into the set of polynomials. Results from earlier questions that you did not succeed in solving or did not have time to prove can still be assumed to hold in the later questions.

If you get really stuck you can ask questions on the project's Zulip stream, in particular for Section 2 on metaprogramming. (And if you have trouble accessing Zulip, then don't hesitate to send us an email: `matthieu.sozeau@inria.fr`, `yannick.forster@inria.fr`, and `theo.winterhalter@inria.fr`.)

At the start of your project submission, please give us an assessment of your previous experience with Coq or other interactive theorem provers. Your project should also include comments explaining your design choices and difficulties for each question.

1 Multivariate Polynomials

Polynomials of $\mathbb{Z}[(X_i)_{i\in\mathbb{N}}]$ will be represented by the following inductive type:

```
Inductive poly : Type :=
| Cst : Z -> poly
| Poly : poly -> nat -> poly -> poly.
```

`Cst` a represents the constant polynomial of value a. `Poly` p i q represents the composite polynomial $p + X_i \cdot q$.

1.1 Valid Polynomials

A constant polynomial `Cst` a is always valid. A composite polynomial `Poly` p i q is valid if and only if:

- all the variables X_j found in p satisfy $j > i$,
- all the variables X_j found in q satisfy $j \geq i$,
- q is not `Cst` 0,
- p and q are valid recursively.

Note: In the following, you may use the function `Nat.compare : nat -> nat -> comparison` to compare two natural numbers or functions like `Nat.ltb` and `Nat.leb` that are boolean checks corresponding to $(<)$ and $(\leq)$ respectively.

a. Define an inductive predicate characterizing the validity of a polynomial.

b. Define a boolean predicate characterizing the validity of a polynomial (`valid_bool` p `= true` when p is valid) and prove it is equivalent to the inductive one.

This predicate can be embedded into a polynomial in order to express valid polynomials:

```
Record valid_poly : Type :=
  { VP_value : poly ;
    VP_prop : valid_bool VP_value = true }.
```

c. Prove that two valid polynomials with the same structure are equal with respect to Leibniz' equality:

$$\forall p, q : \texttt{valid\_poly},\ \texttt{VP\_value}\ p = \texttt{VP\_value}\ q \Rightarrow p = q.$$

Hint: one can start from the dependent elimination of the equality to prove that, for any boolean b, all the proofs of $b = \mathit{true}$ are actually equal. It is the so-called *proof irrelevance*.

1.2 Coefficients and Values

a. Define a type `mono` that can represent any monomial $\prod_{i\in\mathbb{N}} X_i^{\alpha_i}$.
Define a function `get_coef : valid_poly -> mono -> Z` that returns the coefficient of a polynomial associated to a monomial.

b. Prove that two valid polynomials with same coefficients are equal:

$$\forall p, q : \texttt{valid\_poly},\ (\forall m,\ \texttt{get\_coef}\ p\ m = \texttt{get\_coef}\ q\ m) \Rightarrow p = q.$$

c. Define a function that takes a polynomial and a valuation of the variables (that is, a map from X_i to $\mathbb{Z}$) and returns the corresponding value of the polynomial.

d. Prove that two valid polynomials that have the same values have the same coefficients.

1.3 Arithmetic Operations

a. Define a function that computes the sum of two valid polynomials. Prove that this function is a morphism for the evaluation.

b. Define a function that computes the product of two valid polynomials. Prove that this function is a morphism for the evaluation.

2 Boolean Tautologies

2.1 Boolean Formulas

a. Define an inductive type that represents boolean formulas that contain constants $\bot$ and $\top$, variables $v_{i\in\mathbb{N}}$, and boolean operators $\wedge$, $\vee$, $\neg$, ...

b. Define a function that takes a boolean formula (represented by an inductive object) and a valuation of the variables $(v_i)_{i\in\mathbb{N}}$ and returns the boolean result that would be obtained by evaluating the formula.

c. Write a meta-program that takes a Coq expression on booleans and returns the inductive object that represents it and a valuation that contains all the open variables and uninterpreted terms. The meta-program can be a tactic, a MetaCoq `TemplateProgram`, or even a Coq-Elpi program or an Ltac 2 tactic.

Note that the meta-program should behave like the inverse function of the boolean evaluation. In particular, applying the evaluation function to the result of the tactic should be convertible to the original formula.

We show two examples: One in Ltac and one in MetaCoq. Both meta-programs take a term and a partial valuation and return an inductive object and a new valuation that extends the partial one. This example handles only boolean negation. You can extend it to handle all the other cases.

MetaCoq example

```
From MetaCoq.Template Require Import All Checker.
Require Import List.
Import MCMonadNotation.
Open Scope bs.
Open Scope list.

Inductive BTerm :=
| BTnot : BTerm -> BTerm
| BTvar : nat -> BTerm .

Fixpoint index {A} (d : A -> A -> bool) a l :=
  match l with
  | [] => None
  | x :: l =>
      if d x a then Some 0 else
      match index d a l with
      | Some n => Some (S n)
      | None => None
      end
  end.

Definition list_add {A} (d : A -> A -> bool) a l :=
  match index d a l with
  | Some n => (n, l)
  | None => (length l, l ++ [a])
  end.

Fixpoint read_term f l : TemplateMonad (BTerm * list bool) :=
  match f with
  | tApp (tConst cst []) [x] =>
      if eqb cst (MPfile ["Datatypes"; "Init"; "Coq"], "negb")
      then
        mlet (x', l') <- read_term x l ;; ret (BTnot x', l')
      else
        b <- tmUnquoteTyped bool f ;;
        let '(n, l') := list_add (fun b1 b2 => eqb b1 b2) b l in
        ret (BTvar n, l')
  | _ =>
      b <- tmUnquoteTyped bool f ;;
      let '(n, l') := list_add (fun b1 b2 => eqb b1 b2) b l in
      ret (BTvar n, l')
  end.

MetaCoq Run (
  read_term <% ( negb ( negb true )) %> ( @nil bool ) >>=
  tmEval cbv >>=
  tmPrint
).
(* (BTnot (BTnot (BTvar 0)), [true]) *)
```

Ltac example

```
From Coq Require Import List.
Import ListNotations.

Inductive BTerm :=
| BTnot : BTerm -> BTerm
| BTvar : nat -> BTerm.

Ltac list_add a l :=
  let rec aux a l n :=
    match l with
    | []         => constr:((n, a :: l))
    | a :: _    => constr:((n, l))
    | ?x :: ?l =>
      match aux a l (S n) with
      | (?n, ?l) => constr:((n, x :: l))
      end
    end in
  aux a l O.

Ltac read_term f l :=
  match f with
  | negb ?x =>
    match read_term x l with
    | (?x', ?l') => constr:((BTnot x', l'))
    end
  | _ =>
    match list_add f l with
    | (?n, ?l') => constr:((BTvar n, l'))
    end
  end.
```

```
Goal forall a : bool, True.
  intros a.
  let v := read_term (negb (negb a)) (@nil bool) in
  idtac v.
  (* -> (BTnot (BTnot (BTvar 0)), [a]) *)
Abort.
```

2.2 Polynomial Reflection

The goal is to write an automatic tactic for proving that two booleans formulas are equal, by converting them to polynomials over $\mathbb{Z}$ and checking that both polynomials are equal. If you did not succeed in writing the arithmetic operators of Section 1.3, you can use expressions on `Z` instead of `valid_poly` and let the `ring` tactic perform the computations in the following questions.

a. Define a function that transforms a formula represented by an inductive object into a valid polynomial over $\mathbb{Z}$:

- $\overline{a \wedge b} = \overline{a} \times \overline{b}$,
- $\overline{a \vee b} = \overline{a} + \overline{b} - \overline{a} \times \overline{b}$,
- $\overline{\neg a} = 1 - \overline{a}$.

Prove that evaluating a boolean formula f gives the same result than evaluating its polynomial transformation $\overline{f}$.

b. Deduce a process for automatically proving boolean tautologies in Coq. Test it on various boolean equalities, e.g. $\neg a \vee \neg b \vee (c \wedge \top) = \neg(a \wedge b \wedge \neg c)$.

c. How complete is the process? (That is, are there actual boolean equalities that cannot be proved by your tactic?) If so, how can this shortcoming be avoided?