Generating Case Analysis Principles for Inductive Types using MetaCoq

Marcel Ullrich

Advisor - Yannick Forster

Saarland University

Programming Systems Lab

8. November 2019

Case analysis

$$\forall n^{\mathbb{N}}, p\ n$$

$$p\ O \qquad \qquad \forall m, p\ (S\ m)$$

$$m : \mathbb{N} ::= \quad O \quad \mid \quad S\ m$$

Scheme Command

```
Scheme nat_caset := Elimination for ℕ Sort Type.
Print nat_caset.
```

```
nat_caset =
λ (P : ℕ → Type) (f : P 0)
  (f0 : ∀ n : ℕ, P (S n)) ⇒
fix F (n : ℕ) : P n :=
  match n as n0 return (P n0) with
  | 0 ⇒ f
  | S n0 ⇒ f0 n0
  end
     : ∀ P : ℕ → Type,
       P 0 →
       (∀ n : ℕ, P (S n)) →
       ∀ n : ℕ, P n
```

```
MetaCoq Run Scheme nat_caset2 := Elimination for ℕ Sort Type.
Print nat_caset2.
```

```
nat_caset2 =
λ (p : ℕ → Type) (H_O : p 0)
  (H_S : ∀ H : ℕ, p (S H)) ⇒
fix f (inst : ℕ) : p inst :=
  match inst as inst0 return (p inst0) with
  | 0 ⇒ H_O
  | S x ⇒ H_S x
  end
     : ∀ p : ℕ → Type,
       p 0 →
       (∀ H : ℕ, p (S H)) →
       ∀ inst : ℕ, p inst
```

METACOQ

```
Inductive term : Set :=
    tRel : ℕ → term
  | tSort : universe → term
  | tProd : name → term
    → term → term
  | tLambda : name → term
    → term → term
  | tApp : term → ℒ term → term
  | tConst : kername →
    universe_instance → term
  | tInd : inductive →
    universe_instance → term
  | tConstruct : inductive → ℕ
    → universe_instance → term
  | tCase : inductive ∗ ℕ
    → term → term
    → ℒ (ℕ ∗ term) → term
  | tFix : mfixpoint term → ℕ → term
  ...
```

$$\lambda(x : \mathbb{N}).\ x + 0$$

```
tLambda (nNamed "x")
(tInd {|
        inductive_mind := "nat";
        inductive_ind := 0
        |}
    [])
(tApp (tConst "add" [])
      [tRel 0;
      tConstruct {|
        inductive_mind := "nat";
        inductive_ind := 0
        |}
        0
        []
    ])
```

MetaCoq Inductive

```
Record one_inductive_body := {
    ind_name : ident;
    ind_type : term;
    ind_kelim : ℒ sort_family;
    ind_ctors : ℒ (ident * term * ℕ );
    ind_projs : ℒ (ident * term )
}.

Record mutual_inductive_body := {
    ind_npars : ℕ;
    ind params : context;
    ind_bodies : ℒ one_inductive_body;
    ind_universes : universe_context
}.
```

Metaprogramming

Elpi (λProlog):

Enrico Tassi, *Deriving proved equality tests in Coq-elpi: Stronger induction principles for containers in Coq*, 2019.

Typing

$$\texttt{wf}\ \Sigma \rightarrow \texttt{declared\_inductive}\ \Sigma\ mdecl\ ind\ decl \rightarrow$$
$$\exists T, \Sigma; \Gamma \vdash \texttt{createDestruct}\ ind : T$$

MetaCoq

```
Inductive ℕ : Set := O : ℕ | S : ℕ → ℕ
```

Quote ↑

```
{|
ind_finite := Finite;
ind_npars := 0;
ind_params := [];
ind_bodies := [{|
               ind_name := "nat";
               ind_type := tSort (Universe.make'' (Level.lSet, false) []);
               ind_kelim := [InProp; InSet; InType];
               ind_ctors := [("O", tRel 0, 0); ("S", tProd nAnon (tRel 0) (tRel 1), 1)];
               ind_projs := [] |}];
ind_universes := Monomorphic_ctx (LevelSetProp.of_list [], ConstraintSet.empty) |}
{|
mind_entry_record := None;
mind_entry_finite := Finite;
mind_entry_params := [];
mind_entry_inds := [{|
                    mind_entry_typename := "nat";
                    mind_entry_arity := tSort (NEL.sing (Level.lSet, false));
                    mind_entry_template := false;
                    mind_entry_consnames := ["O"; "S"];
                    mind_entry_lc := [tRel 0; tProd nAnon (tRel 0) (tRel 1)] |}];
mind_entry_universes := Monomorphic_ctx
                          ({| LevelSet.this := []; LevelSet.is_ok := LevelSet.Raw.empty_ok |},
                          {| ConstraintSet.this := []; ConstraintSet.is_ok := ConstraintSet.Raw.empty_ok |});
mind_entry_private := None |}
```

Transform →

```
(tLambda (nNamed "p")
   (tProd (nNamed "inst")
      (tInd {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |} [])
      (tSort (NEL.sing (Level.Level "Top.1146", false))))
   (tLambda (nNamed "H_O")
      (tApp (tRel 0)
         [tConstruct {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |} 0 []])
      (tLambda (nNamed "H_S")
         (tProd nAnon (tInd {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |} [])
            (tApp (tRel 2)
               [tApp
                  (tConstruct {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |} 1
                     []) [tRel 0]]))
         (tFix
            [{|
             dname := nNamed "f";
             dtype := tProd (nNamed "inst")
                          (tInd {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |} [])
                          (tApp (tRel 3) [tRel 0]);
             dbody := tLambda (nNamed "inst")
                          (tInd {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |} [])
                          (tCase
                             ({| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |}, 0)
                             (tLambda (nNamed "inst")
                                (tInd
                                   {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |}
                                   []) (tApp (tRel 5) [tRel 0])) (tRel 0)
                             [(0, tRel 3);
                             (1,
                             tLambda nAnon
                               (tInd
                                  {| inductive_mind := "Coq.Init.Datatypes.nat"; inductive_ind := 0 |}
                                  []) (tApp (tRel 3) [tRel 0]))]);
             rarg := 0 |}] 0))))
```

Unquote ↓

```
λ (p : ℕ → Type) (H_O : p 0) (H_S : ∀ H : ℕ, p (S H)) ⇒
fix f (inst : ℕ) : p inst :=
  match inst as inst0 return (p inst0) with
  | 0 ⇒ H_O
  | S x ⇒ H_S x
  end
    : ∀ p : ℕ → Type, p 0 → (∀ H : ℕ, p (S H)) → ∀ inst : ℕ, p inst
```

INDEXED TYPES

$$\frac{}{le\ n\ n}\ le\_n \qquad\qquad \frac{le\ n\ m}{le\ n\ (Sm)}\ le\_S$$

$$\begin{aligned}E_{\leq} : \forall(n : \mathbb{N}),\forall(p : \forall m, n \leq m \rightarrow \mathbb{P}),\\ p\ n\ (le\_n\ n) \rightarrow \\ (\forall m\ (h : le\ n\ m),\ p\ (S\ m)\ (le\_S\ n\ m\ h)) \rightarrow \\ \forall m\ (x : le\ n\ m),\ p\ m\ x\end{aligned}$$

$$\begin{aligned}T &: T_{I_0} \rightarrow \ldots \rightarrow T_{I_k} \rightarrow \mathbb{T} \\ E_T &: \forall(p : \forall I_0 \ldots I_l, T\ I_0 \ldots I_l \rightarrow \mathbb{P}), \\ &\quad p\ i_0 \ldots i_l\ (c_0) \rightarrow \ldots \rightarrow \\ &\quad (\forall a_0 \ldots a_n, p\ i_0 \ldots i_l\ (c_m\ a_0 \ldots a_n)) \rightarrow \\ &\quad \forall I_0 \ldots I_l, \forall(x : T\ I_0 \ldots I_l), p\ I_0 \ldots I_l\ x\end{aligned}$$

PROOF

```
E≤ := λ (n : ℕ)
(p : ∀ m : ℕ, n ≤ m → ℙ)
(H_le_n : p n (le_n n))
(H_le_S : ∀ (m : ℕ) (H : n ≤ m), p (S m) (le_S n m H)).
fix f (m : ℕ) (x : n ≤ m).
match x return (p m x) [
| le_n ⇒ H_le_n
| le_S m x ⇒ H_le_S m x
]
```

Difficulties

$E_T :=$

$\lambda P_0 \dots P_k.$

$\lambda (p : \forall I_0 \dots I_l, T\ P_0 \dots P_k\ I_0 \dots I_l \to \mathbb{P}).$

$\lambda H_0 \dots H_m.$

$\mathit{fix}\ f\ I_0 \dots I_l\ (x : T\ P_0 \dots P_k\ I_0 \dots I_l).$

$\text{match}\ x\ \text{return}\ p\ I_0 \dots I_l\ x\ \text{with}$

$\quad c_i\ a_0 \dots a_n \Rightarrow H_i\ a_0 \dots a_n$

$\quad \dots$

fold over parameter list
modified type of inductive type

- remove parameters
- construct instance using constructor
- construct call to p with indices and instance

take indices and instance
construct match type from indices and instance
quantify real arguments and apply case

Future Work

- Correctness proof for case analysis and induction principles
- Stronger induction principles for nested inductive types:

```
Inductive roseTree := tree (xs:ℒ roseTree).

Scheme Induction for roseTree Sort 𝕋.
∀ P : roseTree → ℙ,
  (∀ xs : ℒ roseTree, P (tree xs)) →
  ∀ r : roseTree, P r
```

```
MetaCoq Run Scheme Induction for roseTree Sort 𝕋.
∀ P:roseTree → ℙ,
  (∀ xs: ℒ roseTree, (∀t, In t xs → P t) → P (tree xs)) →
  ∀ r:roseTree, P r
```

References I

- Abhishek Anand, Simon Boulier, Cyril Cohen, Matthieu Sozeau, and Nicolas Tabareau, *Towards certified meta-programming with typed Template-Coq*, International Conference on Interactive Theorem Proving, Springer, 2018.

- Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter, *The MetaCoq Project*.