Verfication of a Case Analysis Plugin in MetaCoq

Marcel Ullrich

Advisor - Yannick Forster

Saarland University

Programming Systems Lab

13. März 2020

MetaCoq

- framework implementing Coq's logic
- monadic manipulation for plugins
- TemplateCoq implementation: bare type system
- pCUIC implementation: more abstract with clear semantic
- typing is provable
- verfication using meta proofs

MetaCoq

```
{|
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))))
```

Quote ↑

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

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
```

MetaCoq

typing $\Sigma;\Gamma \vdash t : T$

```
Inductive typing Σ Γ : term → term → 𝕋 :=
| type_Rel : ...
| type_Sort : ...
| type_Prod : ...
| type_Lambda : ...
| type_App : ...
| type_Ind : ...
| type_Construct : ...
| type_Case : ...
| type_Fix : ...
| type_Conv : ...
| ...
```

typingSpine $\Sigma;\Gamma \vdash t^T\ xs : T_2$

CHANGES

- changed plugin to use predefined functions
- implementation of induction
- plugin simplification

Correctness

```
wf Σ Γ →
on_ind_body mind mind_body ind ind_body →
declared_inductive Σ mind_body ind ind_body →
  ∀ T t name,
  createElim ind = Some (t,name ) →
  createElimType ind = Some T →
  Σ ; Γ ⊢ t :  T.
```

- concrete type to avoid existentials

Proof structure

1. intros
2. typing of p
3. intros all cases
4. resolve fixpoint
5. wf_local of environment
6. transform to lambda and type to prod
7. intros
8. typing of instance
9. simplification
10. η-Conversion
11. case analysis typing

```
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)(inst : n ≤ m)
   : p m inst :=
match inst as x
   return (p m x) with
| le_n ⇒ H_le_n
| le_S m x ⇒ H_le_S m x
end
```

Auxiliary lemma groups

- function properties and use cases
- basic lemmas
- typing of parts and types
- wf_local and wf_local_rel of environments
- equation for simplification

THE CASE CONSTRUCT

match m	tCase (ℕ,0)	type and parameter count
as n **in** P n **with**	tLambda "n"(tIndℕ) (tApp (tRel 4) [tRel 0]))	return type
	(tRel 0)	match object
| O ⇒ P_0	[(0, tRel 2);	branches
| S n ⇒ P_S n	(1, tLambda "n"(tIndℕ) (tApp (tRel 2) [tRel 0]))	

end

The case rule

```
∀ (indnpar : inductive × ℕ )(u : universe_instance )(p c : term )
(brs : ℒ (ℕ × term ) )(args : ℒ term ),
let ind := indnpar.1 in
let npar := indnpar.2 in
∀ (mdecl : mutual_inductive_body )(idecl : one_inductive_body ),
declared_inductive Σ .1 mdecl ind idecl →  ind_npars mdecl = npar →
let params := firstn npar args in ∀ (ps : universe )(pty : term ),
build_case_predicate_type ind mdecl idecl params u ps = Some pty →
Σ;Γ⊢ p : pty →
existsb (leb_sort_family (universe_family ps ) )(ind_kelim idecl )→
Σ;Γ⊢ c : mkApps (tInd ind u )args →
∀ btys : ℒ (ℕ × term ),
map_option_out (build_branches_type ind mdecl idecl params u p )= Some btys →
All2 (λ br bty : ℕ× term ⇒ (br.1 = bty.1 × Σ;Γ⊢ br.2 : bty.2 )×
Σ;Γ⊢ bty.2 : tSort ps )brs btys →
Σ;Γ⊢ tCase indnpar p c brs : mkApps p (skipn npar args ++ [c] )
```

DIFFICULTIES

- confusing lifting with indices
- search feature often does not work
- problems with unification
- no overview over the goal

Assumption

$$\Sigma; \Gamma, \texttt{params}, \texttt{p}, \texttt{cases}, \uparrow^{1+|\text{ctors}|} \texttt{indices} \vdash$$

$$t^{\text{ind\_type}}$$

$$(\uparrow^{1+|\text{ctors}|}_{|\text{indices}|} \uparrow^{|\text{indicces}|} \texttt{mkRel params} +\!\!+$$

$$\uparrow^{1+|\text{ctors}|}_{|\text{indices}|} \texttt{mkRel indices}) :$$

$$\texttt{tSort x}$$

Goal

$\Sigma; \Gamma, \texttt{params}, \texttt{p}, \texttt{cases}, \texttt{f}, \uparrow^{2+|\text{ctors}|} \texttt{indices} \vdash$

$\uparrow t^{\text{ind\_type}}$

$(\uparrow^{2+|\text{ctors}|+|\text{indicces}|}_{|\text{indices}|} \texttt{mkRel params} +\!\!+$

$\uparrow^{1+|\text{ctors}|}_{|\text{indices}|} \texttt{mkRel indices}) :$

$\texttt{tSort ?s}$

Idea

$$\Sigma; \Gamma, \texttt{xs} \vdash \texttt{t}^{a}\ x : \texttt{T} \rightarrow$$
$$\Sigma; \Gamma, \texttt{ys}, \uparrow^{|ys|}\ \texttt{xs} \vdash$$
$$t^{\uparrow^{|ys|}_{|xs|} a}\ \left(\uparrow^{|ys|}_{|xs|}\ \texttt{x}\right) : \uparrow^{|ys|}_{|xs|}\ \texttt{T}$$

```
apply typing_spine_lifting.
apply t0.
```

REALITY

```
instantiate (1 := x ).
rewrite appLen, lift_context_len.
cbn.
rewrite appLen, revLen, quantifyCasesLen.
rewrite mapLen.
cbn.
rewrite ind_arity_eq.
rewrite removeArityCount.
2: eapply indParamVass;eassumption.
2: now rewrite uniP, nparCount.
rewrite collectProdMkProd.
2: apply indicesVass.
2: cbn;congruence.
rewrite revLen.
unfold snoc, app_context.
unfold snoc, app_context in t0.

rewrite ← appCons.
rewrite ← liftContextSucc.
set (xs:=lift_context ... ).
replace ((lift0 ... )
 (subst_instance_constr univ ... ) ) with
 (lift 1 #|xs| (subst_instance_constr ... ) ).
2: {
  subst xs.
  apply liftSubstInstanceConstr2.
}
change (tSort x ) with (lift 1 #|xs| (tSort x ) ).
replace (mapi
```

```
 (λ (i : ℕ ) _ ⇒
  tRel (...   ) )
 (ind_params mind_body ) ++
map (lift (...   ) (#|ind_indices| ) )
(nat_rec [] ( λ (n : ℕ ) (a : ℒ  term ) ⇒  tRel n :: a ) #|i
  with
    (map (lift 1 #|xs|  )
    (mapi
      (λ (i : ℕ ) _ ⇒
       lift ...   #|ind_indices|
         (( lift0 ...  ) (tRel ...   ) ) )
         (ind_params mind_body ) ++
    map (lift ...   #|ind_indices| )
      (nat_rec [] ( λ (n : ℕ ) (a : ℒ  term ) ⇒
tRel (0 + n ) :: a )
                   #|ind_indices| ) ) ).
2: {  ...   }
subst xs.
pose proof typingSpineLifting.
unfold app_context in X.
apply X.
2: reflexivity.
rewrite uniP, nparCount, sub_diag.
unfold skipn.
rewrite replaceUnderItMkProd.
2: apply indicesVass.
rewrite nparCount in t0.
apply t0.
```

Long goals

```
Σ : global_env_ext
Γ : context
mind_body : mutual_inductive_body
ind: inductive
univ: universe_instance
ind_body : one_inductive_body
Hwf: wf_local Σ Γ
onInd: on_inductive (lift_typing typing ) Σ (inductive_mind ind ) mind_body
onIndB: on_ind_body (lift_typing typing ) Σ (inductive_mind ind ) mind_body (inductive_ind ind ) ind_body
declM: declared_minductive Σ (inductive_mind ind ) mind_body
declI: declared_inductive Σ mind_body ind ind_body
uniP: getParamCount ind_body (ind_npars mind_body ) = ind_npars mind_body
nparCount: ind_npars mind_body = #|ind_params mind_body|
nameF: name → term → name
findBody: nth_error (Ast.ind_bodies mind_body ) (inductive_ind ind ) = Some ind_body
onInductives: Alli (on_ind_body (lift_typing typing ) Σ (inductive_mind ind ) mind_body ) 0 (ind_bodies mind_body )
onParams: on_context (lift_typing typing ) Σ (ind_params mind_body )
onNpars: context_assumptions (ind_params mind_body ) = ind_npars mind_body
onGuard: TemplateTyping.ind_guard mind_body
ind_indices: context
ind_sort: universe
ind_arity_eq: ind_type ind_body =
              it_mkProd_or_LetIn (ind_params mind_body )
                (it_mkProd_or_LetIn ind_indices (TemplateTerm.tSort ind_sort ) )
onArity: on_type (lift_typing typing ) Σ [] (ind_type ind_body )
ind_ctors_sort: ℒ universe
onConstructors: on_constructors (lift_typing typing ) Σ mind_body (inductive_ind ind ) ind_body
                  (ind_ctors ind_body ) ind_ctors_sort
onProjections: ind_projs ind_body <> [] →
                 on_projections (lift_typing typing ) Σ (inductive_mind ind ) mind_body
                   (inductive_ind ind ) ind_body ind_indices
ind_sorts: check_ind_sorts (lift_typing typing ) Σ (ind_params mind_body ) (ind_kelim ind_body ) ind_indices
             ind_ctors_sort ind_sort
== == == == == == == == == == == == == ==
Σ ;Γ
⊢ it_mkLambda_or_LetIn
    (skipn (Ast.ind_npars mind_body − getParamCount ind_body (Ast.ind_npars mind_body ) )
       (Ast.ind_params mind_body ) )
    (tLambda (nNamed "p"%string )
       (replaceLastProd
          (tProd (nNamed "inst"%string )
             (createAppChain
                (( lift0
                    (count_prods
                       (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) ) )
                   (mkApps (tInd ind univ )
                      (mapi
                         (λ (i : ℕ ) (_ : Ast.context_decl ) ⇒
                          tRel (getParamCount ind_body (Ast.ind_npars mind_body ) − i − 1 ) )
                         (skipn (Ast.ind_npars mind_body − getParamCount ind_body (Ast.ind_npars mind_body ) )
                            (Ast.ind_params mind_body ) ) ) ) )
                (count_prods
                   (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) ) )
             (getMaxElim (Ast.ind_kelim ind_body ) ) )
          (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) )
       (it_mkLambda_or_LetIn
          (rev
             (quantifyCases false
                (map
                   (on_snd
                      (Init.Nat.add
                         (Ast.ind_npars mind_body − getParamCount ind_body (Ast.ind_npars mind_body ) ) ) )
                   (Ast.ind_ctors ind_body ) ) (getParamCount ind_body (Ast.ind_npars mind_body ) ) ind univ ) )
          (tFix
             [{|
               dname := nNamed "f"%string;
               dtype := replaceLastProd'
                          (( lift0 (S #|Ast.ind_ctors ind_body| ) )
                             (replaceLastProd
                                (tProd (nNamed "inst"%string )
                                   (createAppChain
                                      (( lift0
                                          (count_prods
                                             (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                (Ast.ind_type ind_body ) ) ) )
                                         (mkApps (tInd ind univ )
                                            (mapi
                                               (λ (i : ℕ ) (_ : Ast.context_decl ) ⇒
                                                tRel (getParamCount ind_body (Ast.ind_npars mind_body ) − i − 1 ) )
                                               (skipn
                                                  (Ast.ind_npars mind_body −
                                                   getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                  (Ast.ind_params mind_body ) ) ) ) )
                                      (count_prods
                                         (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                            (Ast.ind_type ind_body ) ) ) ) (getMaxElim (Ast.ind_kelim ind_body ) ) ) )
                                (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                   (Ast.ind_type ind_body ) ) ) )
                          (createAppChain
                             (tRel
                                (count_prods
                                   (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                      (Ast.ind_type ind_body ) ) + S #|Ast.ind_ctors ind_body| ) )
                             (S
                                (count_prods
                                   (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                      (Ast.ind_type ind_body ) ) ) ) );
               dbody := replaceLastLambda'
                          (mapProdToLambda
                             (( lift0 (S (S #|Ast.ind_ctors ind_body| ) ) )
                                (replaceLastProd
                                   (tProd (nNamed "inst"%string )
                                      (createAppChain
                                         (( lift0
                                             (count_prods
                                                (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                   (Ast.ind_type ind_body ) ) ) )
                                            (mkApps (tInd ind univ )
                                               (mapi
                                                  (λ (i : ℕ ) (_ : Ast.context_decl ) ⇒
                                                   tRel
                                                     (getParamCount ind_body (Ast.ind_npars mind_body ) − i − 1 ) )
                                                  (skipn
                                                     (Ast.ind_npars mind_body −
                                                      getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                     (Ast.ind_params mind_body ) ) ) ) )
                                         (count_prods
                                            (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                               (Ast.ind_type ind_body ) ) ) ) (getMaxElim (Ast.ind_kelim ind_body ) ) )
                                   (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                      (Ast.ind_type ind_body ) ) ) ) )
                          (tCase (ind, Ast.ind_npars mind_body )
                             (( lift0
                                 (S
                                    (count_prods
                                       (remove_arity (Ast.ind_npars mind_body ) (Ast.ind_type ind_body ) ) ) ) )
                                (replaceLastLambda'
                                   (mapProdToLambda
                                      (lift (S (S #|Ast.ind_ctors ind_body| ) )
                                         (Ast.ind_npars mind_body −
                                          getParamCount ind_body (Ast.ind_npars mind_body ) )
                                         (remove_arity
                                            (Ast.ind_npars mind_body −
                                             getParamCount ind_body (Ast.ind_npars mind_body ) )
                                            (replaceLastProd
                                               (tProd (nNamed "inst"%string )
                                                  (createAppChain
                                                     (( lift0
                                                         (count_prods
                                                            (remove_arity
                                                               (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                               (Ast.ind_type ind_body ) ) ) )
                                                        (mkApps (tInd ind univ )
                                                           (mapi
                                                              (λ (i : ℕ ) (_ : Ast.context_decl ) ⇒
                                                               tRel
                                                                 (getParamCount ind_body
                                                                    (Ast.ind_npars mind_body ) − i − 1 ) )
                                                              (skipn
                                                                 (Ast.ind_npars mind_body −
                                                                  getParamCount ind_body
                                                                    (Ast.ind_npars mind_body ) )
                                                                 (Ast.ind_params mind_body ) ) ) ) )
                                                     (count_prods
                                                        (remove_arity
                                                           (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                           (Ast.ind_type ind_body ) ) ) )
                                                  (getMaxElim (Ast.ind_kelim ind_body ) ) )
                                               (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                  (Ast.ind_type ind_body ) ) ) ) ) ) )
                                   (createAppChain
                                      (tRel
                                         (S
                                            (count_prods
                                               (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                  (Ast.ind_type ind_body ) ) + 1 + #|Ast.ind_ctors ind_body| ) ) )
                                      (S
                                         (count_prods
                                            (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                               (Ast.ind_type ind_body ) ) ) ) ) ) ) ) (tRel 0 )
                             (mapi
                                (λ (i : ℕ ) (x : (ident × term ) × ℕ ) ⇒
                                 let (y, count ) := x in
                                 let (_, a ) := y in
                                 (count,
                                 replaceLastLambda'
                                   (( lift0
                                       (S
                                          (count_prods
                                             (remove_arity (Ast.ind_npars mind_body ) (Ast.ind_type ind_body ) ) ) ) )
                                      (lift (S (S #|Ast.ind_ctors ind_body| ) )
                                         (Ast.ind_npars mind_body −
                                          getParamCount ind_body (Ast.ind_npars mind_body ) )
                                         (( mapProdToLambda (remove_arity (Ast.ind_npars mind_body ) a ) )
                                          {Ast.ind_npars mind_body := tInd ind univ} ) ) )
                                   (createAppChain
                                      (createAppChainOff
                                         (S
                                            (count_prods
                                               (remove_arity (Ast.ind_npars mind_body ) (Ast.ind_type ind_body ) ) +
                                             count ) )
                                         (tRel
                                            (S
                                               (count_prods
                                                  (remove_arity
                                                     (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                                     (Ast.ind_type ind_body ) ) + count +
                                                (#|Ast.ind_ctors ind_body| − i ) ) ) ) )
                                         (Ast.ind_npars mind_body −
                                          getParamCount ind_body (Ast.ind_npars mind_body ) ) ) count ) ) )
                                (Ast.ind_ctors ind_body ) ) );
               rarg := count_prods
                         (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                            (Ast.ind_type ind_body ) ) |}] 0 ) ) )
  : Ast.it_mkProd_or_LetIn
      (skipn (Ast.ind_npars mind_body − getParamCount ind_body (Ast.ind_npars mind_body ) )
         (Ast.ind_params mind_body ) )
      (tProd (nNamed "p"%string )
         (replaceLastProd
            (tProd (nNamed "inst"%string )
               (createAppChain
                  (( lift0
                      (count_prods
                         (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) ) )
                     (mkApps (tInd ind univ )
                        (mapi
                           (λ (i : ℕ ) (_ : Ast.context_decl ) ⇒
                            tRel (getParamCount ind_body (Ast.ind_npars mind_body ) − i − 1 ) )
                           (skipn (Ast.ind_npars mind_body − getParamCount ind_body (Ast.ind_npars mind_body ) )
                              (Ast.ind_params mind_body ) ) ) ) )
                  (count_prods
                     (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) ) )
               (getMaxElim (Ast.ind_kelim ind_body ) ) )
            (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) )
         (Ast.it_mkProd_or_LetIn
            (rev
               (quantifyCases false
                  (map
                     (on_snd
                        (Init.Nat.add (Ast.ind_npars mind_body − getParamCount ind_body (Ast.ind_npars mind_body ) ) ) )
                     (Ast.ind_ctors ind_body ) ) (getParamCount ind_body (Ast.ind_npars mind_body ) ) ind univ ) )
            (replaceLastProd'
               (( lift0 (S #|Ast.ind_ctors ind_body| ) )
                  (replaceLastProd
                     (tProd (nNamed "inst"%string )
                        (createAppChain
                           (( lift0
                               (count_prods
                                  (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                     (Ast.ind_type ind_body ) ) ) )
                              (mkApps (tInd ind univ )
                                 (mapi
                                    (λ (i : ℕ ) (_ : Ast.context_decl ) ⇒
                                     tRel (getParamCount ind_body (Ast.ind_npars mind_body ) − i − 1 ) )
                                    (skipn
                                       (Ast.ind_npars mind_body − getParamCount ind_body (Ast.ind_npars mind_body ) )
                                       (Ast.ind_params mind_body ) ) ) ) )
                           (count_prods
                              (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) )
                                 (Ast.ind_type ind_body ) ) ) ) (getMaxElim (Ast.ind_kelim ind_body ) ) ) )
                  (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) ) )
               (createAppChain
                  (tRel
                     (count_prods
                        (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) +
                      S #|Ast.ind_ctors ind_body| ) )
                  (S
                     (count_prods
                        (remove_arity (getParamCount ind_body (Ast.ind_npars mind_body ) ) (Ast.ind_type ind_body ) ) ) ) ) ) ) )
```

METACOQ DIFFICULTIES

- deeply nested definitions
- notational inconvenience
- function definitions with fold_left, reverse and mapi
- properties hidden in definitions and lemmas
- missing lemmas for properties of functions
- some lemmas and functions only in pCUIC

Future Work

- Move plugin to pCUIC
- Induction principles for mutual inductive types
- Stronger induction principles for nested inductive types

```
Inductive roseTree := tree (xs:list 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*.

Induction

```
λ(A : 𝕋)(R : A → A → ℙ)(p : ∀ x : A, Acc R x → 𝕋)
(H_Ai : ∀(x : A)(H : ∀ y : A, R y x → Acc R y),
  (∀(y : A)(H_0 : R y x), p y (H y H_0)) →
  p x (Acc_intro x H)),
fix f (x : A)(inst : Acc R x) : p x inst :=
match inst as inst0 return (p x inst0) with
| Acc_intro x_0 ⇒ H_Ai x x_0
    (λ(y : A)(H : R y x), f y (x_0 y H))
end
```

1. find arguments with inductive call as argument
2. extract indices
3. add inductive hypothesis
4. lift everything accordingly

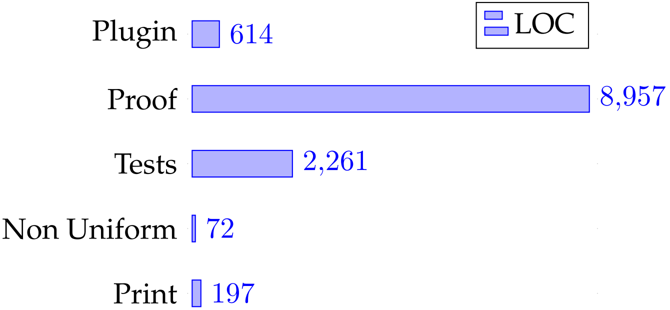
GOALS IN THE WILD

```
typing_spine Σ
  (( vass (nNamed "f"%string )
     (( lift0 0 )
       (replaceLastProd'
         (( lift0 (S #|ind_ctors ind_body| ) )
           (replaceLastProd
             (tProd (nNamed (ind_name ind_body ) )
               (mkApps
                 (( lift0
                    #|collect_prods
                       (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| )
                   (tInd ind univ ) )
                 (mapi
                   ( λ (i : ℕ ) (_ : context_decl ) ⇒
                    (lift0
                      #|collect_prods
                         (remove_arity (getParamCount ind_body (ind_npars mind_body ) )
                           (ind_type ind_body ) )| )
                     (tRel (getParamCount ind_body (ind_npars mind_body ) − i − 1 ) ) )
                   (skipn (ind_npars mind_body − getParamCount ind_body (ind_npars mind_body ) )
                     (ind_params mind_body ) ) ++
                  ℕ_rec (λ _ : ℕ ⇒  ℒ term ) []
                   ( λ (n : ℕ ) (a : ℒ term ) ⇒ tRel (0 + n ) :: a )
                   #|collect_prods
                      (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| ) )
               (getMaxElim (ind_kelim ind_body ) ) )
             (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) ) ) ) )
         (tApp
           (tRel
             (#|collect_prods
                 (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| +
              S #|ind_ctors ind_body| ) )
           (tRel
             #|collect_prods
                (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )|
            :: ℕ_rec (λ _ : ℕ ⇒  ℒ term ) [] (λ (n : ℕ ) (a : ℒ term ) ⇒ tRel n :: a )
               #|collect_prods
                  (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| ) ) ) )
  :: (Γ , skipn (ind_npars mind_body − getParamCount ind_body (ind_npars mind_body ) ) (ind_params mind_body ) ),,
     vass (nNamed "p"%string )
      (replaceLastProd
        (tProd (nNamed (ind_name ind_body ) )
          (mkApps
            (( lift0
               #|collect_prods
                  (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| )
              (tInd ind univ ) )
            (mapi
              ( λ (i : ℕ ) (_ : context_decl ) ⇒
               (lift0
                 #|collect_prods
                    (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| )
                (tRel (getParamCount ind_body (ind_npars mind_body ) − i − 1 ) ) )
              (skipn (ind_npars mind_body − getParamCount ind_body (ind_npars mind_body ) )
                (ind_params mind_body ) ) ++
             ℕ_rec (λ _ : ℕ ⇒  ℒ term ) [] (λ (n : ℕ ) (a : ℒ term ) ⇒ tRel (0 + n ) :: a )
              #|collect_prods
                 (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| ) )
          (getMaxElim (ind_kelim ind_body ) ) )
        (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) ) ),
     rev
      (quantifyCases false
        (map (on_snd (Init.Nat.add (ind_npars mind_body − getParamCount ind_body (ind_npars mind_body ) ) ) )
          (ind_ctors ind_body ) ) (getParamCount ind_body (ind_npars mind_body ) ) ind univ ) ),
   lift_context (1 + S #|ind_ctors ind_body| ) 0 ind_indices )
  (( lift0
     #|lift_context (1 + S #|ind_ctors ind_body| ) 0 ind_indices ++
      vass (nNamed "f"%string )
       (( lift0 0 )
         (replaceLastProd'
           (( lift0 (S #|ind_ctors ind_body| ) )
             (replaceLastProd
               (tProd (nNamed (ind_name ind_body ) )
                 (mkApps
                   (( lift0
                      #|collect_prods
                         (remove_arity (getParamCount ind_body (ind_npars mind_body ) )
                           (ind_type ind_body ) )| ) (tInd ind univ ) )
                   (mapi
                     ( λ (i : ℕ ) (_ : context_decl ) ⇒
                      (lift0
                        #|collect_prods
                           (remove_arity (getParamCount ind_body (ind_npars mind_body ) )
                             (ind_type ind_body ) )| )
                       (tRel (getParamCount ind_body (ind_npars mind_body ) − i − 1 ) ) )
                     (skipn (ind_npars mind_body − getParamCount ind_body (ind_npars mind_body ) )
                       (ind_params mind_body ) ) ++
                    ℕ_rec (λ _ : ℕ ⇒  ℒ term ) []
                     ( λ (n : ℕ ) (a : ℒ term ) ⇒ tRel (0 + n ) :: a )
                     #|collect_prods
                        (remove_arity (getParamCount ind_body (ind_npars mind_body ) )
                          (ind_type ind_body ) )| ) ) (getMaxElim (ind_kelim ind_body ) ) )
               (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) ) ) ) )
           (tApp
             (tRel
               (#|collect_prods
                   (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| +
                S #|ind_ctors ind_body| ) )
             (tRel
               #|collect_prods
                  (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )|
              :: ℕ_rec (λ _ : ℕ ⇒  ℒ term ) [] (λ (n : ℕ ) (a : ℒ term ) ⇒ tRel n :: a )
                 #|collect_prods
                    (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| ) ) ) ) )
      :: rev
          (quantifyCases false
            (map
              (on_snd (Init.Nat.add (ind_npars mind_body − getParamCount ind_body (ind_npars mind_body ) ) ) )
              (ind_ctors ind_body ) ) (getParamCount ind_body (ind_npars mind_body ) ) ind univ ) ++
        [vass (nNamed "p"%string )
          (replaceLastProd
            (tProd (nNamed (ind_name ind_body ) )
              (mkApps
                (( lift0
                   #|collect_prods
                      (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| )
                  (tInd ind univ ) )
                (mapi
                  ( λ (i : ℕ ) (_ : context_decl ) ⇒
                   (lift0
                     #|collect_prods
                        (remove_arity (getParamCount ind_body (ind_npars mind_body ) )
                          (ind_type ind_body ) )| )
                    (tRel (getParamCount ind_body (ind_npars mind_body ) − i − 1 ) ) )
                  (skipn (ind_npars mind_body − getParamCount ind_body (ind_npars mind_body ) )
                    (ind_params mind_body ) ) ++
                 ℕ_rec (λ _ : ℕ ⇒  ℒ term ) []
                  ( λ (n : ℕ ) (a : ℒ term ) ⇒ tRel (0 + n ) :: a )
                  #|collect_prods
                     (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) )| ) )
              (getMaxElim (ind_kelim ind_body ) ) )
            (remove_arity (getParamCount ind_body (ind_npars mind_body ) ) (ind_type ind_body ) ) )]| )
     (subst_instance_constr univ (ind_type ind_body ) ) )
  (mapi
     ( λ (i : ℕ ) (_ : context_decl ) ⇒
      (lift0 (1 + S #|ind_ctors ind_body| + #|ind_indices| ) ) (tRel (ind_npars mind_body − i − 1 ) ) )
     (ind_params mind_body ) ++
   map (lift (1 + S #|ind_ctors ind_body| ) (#|ind_indices| + 0 ) )
     (nat_rec (λ _ : ℕ ⇒  ℒ term ) [] (λ (n : ℕ ) (a : ℒ term ) ⇒ tRel (0 + n ) :: a ) #|ind_indices| ) )
  (tSort ?s2 )
```

USED AXIOMS

- parameters and indices have empty bodys
- fix guard axiom
- induction properties

Lines of Code



Graph