

Certified programming with dependent types made simple with proxy-based small inversions

Pierre Corbineau

Basile Gros

Jean-François Monin

VERIMAG, Univ. Grenoble Alpes, CNRS, Grenoble INP¹

27 September, 2025

¹Institute of Engineering Univ. Grenoble Alpes





Motivation

Proxy-based small inversions
allow for dependent programming
with simplified and readable code.



Examples

- Definition of transposition of size-indexed matrices (vectors of vectors) and proof that this transposition is involutive.
- Manipulation of finite sets **Fin.t**, following a challenging use-case proposed by Clément Pit-Claudel

Small inversion

- The conclusion of the elimination scheme for `Fin.t` is $\forall n, \forall (x:\text{Fin.t } n), P \text{ } n \text{ } x$

- Objective: constrain `n` to be 3 : $\forall (x:\text{Fin.t } 3), P \text{ } x$

- Historical methods change the conclusion:
 $\forall n, \forall (x:\text{Fin.t } n), n = 3 \Rightarrow P \text{ } n \text{ } x.$

- Proxy-based small inversions change the matched objet.

► Create a proxy inductive type that mimics `Fin.t 3`, and can be eliminated without loss of information.

► We go from $(x:\text{Fin.t } 3) \longrightarrow P \text{ } x$
 to $(x:\text{Fin.t } 3) \longrightarrow \text{proxy}(\text{Fin.t } (S \text{ } 2)) \longrightarrow P \text{ } x$

Partial inductive types

- First, *partial inductive types* mimic the comportment of the inductive type when specialised to a given pattern of the index.
- We work with inductive indices, the possible primitive patterns for the index are built from the constructors of its type.

```
Inductive Fin.t : nat → Set :=  
| F1 : ∀ n : nat, Fin.t (S n)  
| FS : ∀ n : nat, Fin.t n → Fin.t (S n).
```

```
Inductive Fin_0 : Set :=.  
Inductive Fin_S (n : nat) : Set :=  
| is_F1 : Fin_S n  
| is_FS (r:Fin.t n) : Fin_S n.
```

Partial inductive types for dependent inversion

For dependent inversion, we also keep trace of the structure of the object we invert.

```
Inductive Fin_0 : Fin.t 0 -> Set :=.
```

```
Inductive Fin_S (n : nat) : Fin.t (S n) -> Set :=
```

```
| is_F1 : Fin_S n F1
```

```
| is_FS (r:Fin.t n) : Fin_S n (FS r).
```

Selecting the inductive type

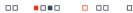
- Then, two translation functions translate the original object into an object of the corresponding partial inductive type.
- The first maps index values to the partial inductive types.

Definition $\text{Fin_proxy_type} (n:\text{nat}) : \text{Fin.t } n \rightarrow \text{Set} :=$
 $\text{match } n \text{ with}$
 | 0 $\Rightarrow \text{Fin}_0$
 | S m $\Rightarrow \text{Fin}_S m$
 end.

Translating the inductive type

The second maps constructors to their proxy counterpart.

```
Definition Fin_proxy{n} (r : Fin.t n) : Fin_proxy_type n r :=  
  match r as r' in Fin.t n' return Fin_proxy_type n' r' with  
  | F1 n      ⇒ is_F1 n  
  | FS n t'   ⇒ is_FS n t'  
end.
```

Using the proxy

- These objects only need to be created once.
- To use them, we then perform an elimination of the translated proxy object:

```
match Fin_proxy x with
| is_F1 _      ⇒ p1
| is_FS _ x' ⇒
    match Fin_proxy x' with
    | is_F1 _      ⇒ p2
    | is_FS _ x'' ⇒ ...
```



Using the proxy : typeclass

It is possible to wrap the proxy in a typeclass so that remembering the proxy name is not necessary.

```
Class Proxy (T:Type) :=  
{ proxy_type: Type;  
  proxy:      T → proxy_type }.
```

```
Class dProxy (T:Type) :=  
{ dproxy_type: T → Type;  
  dproxy:      ∀ t:T, dproxy_type t }.
```

```
match dProxy/proxy (x : Fin.t 3) with ...
```

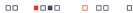


Systematic creation

Partial inductive types and proxies are systematically derived by successive refinements of the inductive type through different transformations:

- Derecursionisation: removing recursive references to the inductive type.
- Deparameterisation: transforming parameters into indices.
- Transformation into dependent inversion *if needed*.
- Specialisation: creating partial inductives for a given inductively typed index;
can be iterated for deep or multiple patterns.
- Parameterisation: transforming as many indices as possible into parameters.





Current and future work

Ongoing work:

- MetaRocq plugin that automates the definition of proxies.
- Exploration of edge cases in the transformations.
- Case studies (CompCert...)

Future objectives:

- Support for inversion with dependently typed indices.
- Support for inversion with non-linear patterns.
- Eventually: integration of proxy-based small inversions into the Equations plugin?

