Small inversions for smaller inversions

Jean-François Monin*

Abstract

We describe recent improvements on *small inversions*, a technique presented earlier as a possible alternative to Coq standard inversion.

Many proofs relying on inductive definitions require so-called *inversion* steps in order to exploit the information contained in a hypothesis $H : Ra_0 \dots a_n$, where R is a dependent inductive relation (or type) applied to actual parameters $a_0 \dots a_n$. In Coq, such steps are usually performed using a powerful tactic called `inversion`. In previous work [MS13], we proposed an alternative lightweight approach, which is not automated (in contrast with Coq `inversion`) but provides a better understanding on what happens, a full control on the proof script and smaller proof terms. In particular, no additional (proof of) equalities are introduced. Moreover, there are situations involving dependent types where standard `inversion` fails whereas small inversion succeeds. In all approaches, inversion is essentially a complex dependent pattern matching on H. In [MS13], this is handled using auxiliary functions.

However, we discovered later that standard `inversion` has an additional important advantage on the previous version of small inversions: it provides syntactically strict subterms of H which can directly be used in recursive calls of a fixpoint definition. In the Braga method designed with D. Larchey-Wendling [LWM18, LM21], we showed how clear and explicit subterms of H can be recovered using projections π_R defined with a dependent pattern matching similar to small inversions – let us call them *smaller inversions*.

We present here an improvement on small inversions, where auxiliary functions are replaced with auxiliary inductive types which are easier to understand and to use. The new small inversion is more powerful: it can handle goals involving terms with occurrences of H. Such goals naturally arise in direct proofs of partial correctness properties of functions – for instance, but not only, fixpoints obtained by the Braga method. Standard `inversion` turns out to be very often unusable there.

As a foretaste, consider a *reference* function for OCaml `fold_left` – efficiency is then irrelevant here – honestly defined by a right to left traversal of its list argument. To this effect we introduce an auxiliary *non-recursive* dependent data type `rl` l with two constructors: `Nilr` of type `rl` $[]$ – reflecting the empty list – and `Consr` of type `rl` $(u +: z)$ – where $u +: z$ is the catenation of a list u and a single element z. Following the Braga method, we first define an inductive domain D_{list} for termination certificates. Here D_{list} contains `Nilr`, as well as `Consr` $u\ z$ whenever `l2r` u, the reflection of u, is itself in D_{list}. Given $d : D_{list}$ (`Consr` $u\ z$) we then define the projection $\pi\ d$ which provides its structurally smaller component of type D_{list} (`l2r` u), allowing us to easily define fixpoints such as `foldl_ref` below, where b_0 and f are respectively an initial value and a function to be folded, and `rew` d is an administrative rewriting step transforming `l2r` $(u +: z)$ into `Consr` $u\ z$.

```
Fixpoint foldl_ref l (d: D_list (l2r l)): B :=
  match l2r l in rl l return D_list (l2r l) → B with
  | Nilr      => λ d, b0
  | Consr u z => λ d, f (foldl_ref u (π (rew d))) z
  end d.
```

Reasoning on such functions commonly requires inversion steps on d. For instance we would like to prove that the actual standard tail-recursive algorithm returns the same result as

* Verimag, Université Grenoble Alpes, CNRS, Grenoble INP

`foldl_ref`. But we already get an issue with a much more elementary fact, stating that for any $d : D_{list}$ `Nilr`, we have `foldl_ref` $[]$ $d = b_0$: this turns out to be out of reach of Coq standard inversion. In [LM21], this issue is circumvented by replacing the former definition of `foldl_ref` by an enriched program which returns an inhabitant of $\{b : B \mid G_{foldl}\ l\ b\}$ instead of just B, where G_{foldl} is a suitable characteristic relation. With our small inversion described below, we can directly reason on `foldl_ref` as defined above. More details and additional examples are available at https://www-verimag.imag.fr/~monin/Proof/Small_inversions/2022 .

For a more general situation, consider an inductive relation $R : T_0 \to T_1 \ldots \to T_n \to Sort$, where $Sort$ is a sort (e.g., Prop or Type). Whatever the technology to be used, the key point is that inversion makes sense when:

- at least one type among $T_0, T_1 \ldots T_n$, say T_0, is itself an inductive type; without loss of generality we consider here that there is exactly one such type; below we write $\mathbf{T}$ for $T_1 \ldots T_n$;
- in the hypothesis H to be inverted, the corresponding actual parameter $a_0 : T_0$ has a specialized shape σ, corresponding to a pattern $C\ args$ starting with a constructor C of T_0 (in many cases, $args$ are just variables).

In general, only a subset of the constructors of R are compatible with the shape of a_0. Inversion then proceeds by *simultaneous* pattern-matching on H and a_0, in order to select the relevant cases of R.

We proceed as follows. For each shape of interest σ we derive from the definition of T_0 an inductive specialized version $T_{0\sigma}$ of T_0. $T_{0\sigma}$ is a copy-paste of the relevant (compatible with σ) constructors of T_0, with appropriate modifications: the variables $x_1 \ldots x_\sigma$ of σ become parameters of $T_{0\sigma}$; the type of $T_{0\sigma}\ x_1 \ldots x_\sigma$ is $\forall\, \mathbf{a} : \mathbf{T}, R\, \sigma\, \mathbf{a} \to Sort$ (it is empty for absurd cases). We then define, by dependent pattern matching on r, the function $R_{inv}\ y_0\ \mathbf{y}$ of type $\forall\, r : R\, y_0\, \mathbf{y}$, $(\texttt{match}\ y_0\ \texttt{with} \ldots \mid \sigma_i \Rightarrow T_{0\sigma_i}\ x_1 \ldots x_{\sigma_i} \mid \ldots \texttt{end})\, r$.
The main argument of R_{inv} is r (its other arguments $y_0\ \mathbf{y}$ will be left implicit). An obvious requirement on the shapes σ_i occuring in the above pattern matching is that they cover T_0. Inverting H is then just a pattern matching on $R_{inv}\ H$, whose type reduces to the relevant $T_{0\sigma_i}$. Possible occurrences of H in the goal are correctly dealt with for free thanks to the additional argument r of $T_{0\sigma}$.

In this version, the components of H are not considered as subterms of H because they are repackaged in a constructor of $T_{0\sigma}$. In the Braga method, where the subterm property is needed, an argument of type $R\, y_0\, \mathbf{y}$ can be added to $T_{0\sigma}$ and its final argument uses of $\pi_R\ \sigma$ instead of σ. Even if the sort of T_0 is Prop, we can then obtain a fully general recursion principle T_0`_rect` – an improvement on [LM21] which is limited to proof irrelevant statements.

References

[LM21] Dominique Larchey-Wendling and Jean-François Monin. *The Braga Method: Extracting Certified Algorithms from Complex Recursive Schemes in Coq*, chapter 8, pages 305–386. World Scientific, September 2021.

[LWM18] Dominique Larchey-Wendling and Jean-François Monin. Simulating Induction-Recursion for Partial Algorithms. In *24th International Conference on Types for Proofs and Programs,TYPES 2018*, Braga, Portugal, June 2018.

[MS13] Jean-François Monin and Xiaomu Shi. Handcrafted Inversions Made Operational on Operational Semantics. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, pages 338–353. Springer, 2013.