Trakt: a generic pre-processing tactic for theory-based proof automation

Denis Cousineau[1], Enzo Crance[1,2], and Assia Mahboubi[2]

[1] Mitsubishi Electric R&D Centre Europe (MERCE), Rennes, France
[2] Inria Rennes Bretagne Atlantique, Nantes, France

Abstract

We present Trakt, a Coq plugin to pre-process goals before handing them to theory-based proof automation tactics. It is designed to be user-friendly, generic and extensible. The plugin is available with documentation and examples on GitHub[1].

Pre-processing and automation Various decision procedures are implemented as tactics constituting a real toolbox for the working Coq user. Some of these tactics focus on a specific theory, such as `lia` [1] for linear integer arithmetic. Others are able to handle a combination of theories, such as the `smt` tactic from the SMTCoq plugin [2] or the `itauto` solver [3]. All of them have a well-defined signature of goals that they can process. However, this signature makes tactics inflexible, as they support a specific set of encodings for a mathematical object, but reject any other encoding, even provably equivalent.

To broaden the input space of theory-based automation tactics, a solution is to introduce a pre-processing phase whose purpose is to reframe the goal with types and values that are part of the signature of the target automation tactic. Several tools were designed with this objective in mind, such as the `zify` tactic [4] aiming to canonise arithmetic and logical goals so that they can be handled by the `lia` tactic. Various integer values (*e.g.* in `nat` or `positive`) are embedded into `Z`, which is the standard integer type recognised by the automation tactic. The tactic offers a typeclass-based way to extend its knowledge database, containing associations between source and target terms. For example, the `mczify` tactic [5] is a set of `zify` typeclass instances specifically tailored to open this tactic to a maximum of goals whose arithmetic content is expressed with data types from the MathComp library [6].

Pre-processing, beyond arithmetic in standard Coq Our plugin, Trakt, behaves as a generic version of `zify`, extended on both the input and output sides. On the input side, its flexibility allows handling uninterpreted symbols and universal quantifiers, which are common in Coq goals and well-known by automation tools like SMT solvers. The knowledge database is filled through user-friendly Coq commands, so that the user does not have to dive into the internals of the plugin, and provides a minimum of terms thanks to the use of inference. Commands are available to declare embeddings between types (*e.g.* embedding `nat` into `Z`[2]), relations (*e.g.* linking `@eq nat` to `Z.eqb`), and symbols (*e.g.* `O` and `Z0` or `Nat.add` and `Z.add`). The tool supports both boolean and propositional logic, meaning it is able to express any decidable predicate in the selected output logic. Pre-processing is then made through the `trakt` tactic. It performs a recursive inspection on the source goal and gradually builds the output goal as well as a correctness proof stating that the output goal implies the input goal. On the output side, the tactic is parameterised and does not target a specific automation tool.

[1] https://github.com/ecranceMERCE/trakt

[2] NB: The embedding from `nat` to `Z` is partial. In this case, our tool adds a positivity hypothesis in the output goal for every value embedded from `nat`.

Indeed, it has been tested on `lia` and successfully integrated into a version of the SMTCoq plugin.

Below is an example of a Coq goal involving propositional logic, arithmetic expressions in the MathComp type `int`, as well as an uninterpreted function in `nat`:

```
forall (f : int -> nat) (x : int),
  x = 4 -> f (2%:Z * x + 1) = f 5.
```

Say the user wants to send this goal to an SMT solver through the SMTCoq plugin. This tool is able to process goals with boolean logic and arithmetic expressed in `Z`. However, it is not the case in the current goal. Here is the result after running `trakt Z bool`:

```
forall (f' : Z -> Z), (forall (z : Z), 0 <=? f' z = true) -> forall (x' : Z),
  x' =? 4 = true -> f' (2 * x' + 1) =? f' 5 = true.
```

As we can see, no value is left in an integer type that is not `Z`, and logic is fully expressed in `bool`. Thanks to our pre-processing phase, the goal can now be solved automatically.

Implementation choices Trakt is implemented in Coq-Elpi [7], a meta-language that has been shown to be relevant on several aspects. First of all, the HOAS encoding [8] of Coq terms allows processing binders without having to worry about De Bruijn indices, and extending the data type representing Coq terms with new constructors in order to annotate nodes temporarily during the execution of the tactic. Moreover, Coq-Elpi is a very high-level meta-language, offering an enjoyable level of abstraction: the association of Coq unification variables to Prolog variables and the (anti-)quotation system allow term construction using Coq notations and implicit arguments, and elaborators are available to infer the type of all binders in Coq terms. In addition, knowledge databases associated to commands and tactics are represented as Prolog data clauses, which makes the queries natural and idiomatic in the logic programming paradigm. Finally, we also have full control on the performance of the algorithm: as opposed to Ltac featuring only syntactic matching and the typeclass system whose inference algorithm is hard to control, Coq-Elpi lets us decide where we want to use Coq conversion, and order the clauses to control how the backtracking is done.

References

[1] Frédéric Besson. Fast reflexive arithmetic tactics the linear case and beyond. In *International Workshop on Types for Proofs and Programs*, pages 48–62. Springer, 2006.

[2] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. SMTCoq: A plug-in for integrating SMT solvers into Coq. In *International Conference on Computer Aided Verification*, pages 126–133. Springer, 2017.

[3] Frédéric Besson. Itauto: An Extensible Intuitionistic SAT Solver. In *12th International Conference on Interactive Theorem Proving (ITP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[4] Frédéric Besson. ppsimpl: a reflexive Coq tactic for canonising goals. CoqPL, 2017.

[5] Kazuhiko Sakaguchi. `mczify`. https://github.com/math-comp/mczify.

[6] Assia Mahboubi and Enrico Tassi. Mathematical components, 2017. https://math-comp.github.io/mcb/.

[7] Enrico Tassi. Elpi: an extension language for Coq (Metaprogramming Coq in the Elpi λProlog dialect). 2018.

[8] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. *ACM sigplan notices*, 23(7):199–208, 1988.