A Decision Procedure for Equivalence Relations

Sébastien Michelland[1], Pierre Corbineau[2], Lionel Rieg[2], and Karine Altisen[2]

[1] ENS de Lyon
sebastien.michelland@ens-lyon.fr
[2] Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG
<First Name>.<Last Name>@univ-grenoble-alpes.fr

Motivation

Equality or equivalence between objects is a typical kind of Coq goal that regularly appears in theories. When it is a consequence of the context, one can simply `rewrite` their way through, but it is more convenient to handle it with a tactic. The `congruence` tactic by Pierre Corbineau [1] is a decision procedure designed to solve these problems.

For instance, given some lists `l1`, `l2` and `l3`, `congruence` will prove the following goal.

```
Goal:  l1 ++ l2 = l3 -> l2 ++ l1 = l3 ->
       (forall x y z, (x ++ y) ++ z = x ++ (y ++ z)) ->
       l1 ++ l3 = l3 ++ l1.
```

The `congruence` tactic uses the congruence closure algorithm by Downey, Sethi and Tarjan [3] to saturate the set of known equalities; this method basically propagates equalities through calls to equal functions. Proofs are tracked along the way in the style of [4]. The tactic is also able to instantiate quantified equalities such as the associativity property of `++`, and use injectivity and discrimination of inductive constructors [2]. The Lean theorem prover also features a congruence-closure based decision procedure which can handle heterogeneous equalities [5].

One of the limits of `congruence` is its restriction to the Coq equality `eq` even for functions or propositions. This is not structural since the congruence closure method applies equally well to any equivalence relation. Using equivalence relations would also prompt the tactic to work with typeclasses and setoids, which are more common in modern Coq code. As an additional benefit, typeclasses register equivalence relations and let the tactic find them automatically.

Generalization to Equivalence Relations

We propose to generalize `congruence` by supporting hierarchies of equivalence relations in the congruence closure algorithm (this is, to the best of our knowledge, a new extension). This new tactic would be used to decide ground relations of the form `R x y` using relations in the context. When universally-quantified relation hypotheses are involved, we can only hope to have a semi-decision procedure.

An instance of relation hierarchy on lists would be

List equality (`=`) $\subseteq$ Multiset equality (`==`) $\subseteq$ Set equality (`===`),

which is modeled by three instances of the `Equivalence` typeclass and two instances of `subrelation`. These instances can be detected automatically through the typeclass mechanism even when not in the current context. This generalization substantially raises the expressiveness, as the tactic can now use (or even discover) properties like

```
forall l1 l2, l1 ++ l2 == l2 ++ l1,
```

and automatically transpose them to weaker relations (in this case, set equality). Because this method supports arbitrary user-defined relations, we expect it to be useful in a wider variety of contexts than `congruence`.

The congruence closure algorithm and its data structures are extended to support multiple relations and proof trees for each term in the system. Universally-quantified hypotheses are instantiated by matching the quantified variables against equivalence classes instead of individual elements, as this reduces the number of redundant matches.

The tactic also works with arbitrary non-reflexive equivalence relations (called partial equivalence relations or PERs), which allows functions to be treated in the same way as the objects they are manipulating. PERs are especially useful to describe morphisms of relations (typically `R x y -> R' (f x) (f y)`) which correspond to the extensional equality found in `congruence`.

A natural extension of this method is the support for reasoning modulo logical equivalence on `Prop`. A proposition that is found equivalent to `True` or `False` for `<->` is in fact decided, and can thus be proven or disproven. This makes it possible to deal with arbitrary propositions as goals (instead of just `R x y`) and prove them equivalent to `True`.

Development of the Tactic

The tactic, still under development, started as a command-line tool written in Ocaml and is currently being integrated into Coq as a plugin. This tool generates standalone Coq proof scripts which can be checked independently. It implements the extended congruence closure algorithm, along with inclusion hierarchies and PERs. Universally-quantified relation hypotheses can be instantiated by the matching algorithm, which makes it possible to describe simple first-order theories.

The main challenges and upcoming work mainly lie in the following directions:

- Design heuristics to control the instantiation of universally-quantified hypotheses (to avoid combinatorial issues).
- Use the Coq context and typeclasses to limit the amount of input from the user.
- Experiment with reasoning modulo logical equivalence.
- Design a test suite that compares the effectiveness and performance of the new tactic to `congruence`.

References

[1] `congruence` tactic in the Coq documentation. https://coq.inria.fr/distrib/V8.11.2/refman/proof-engine/tactics.html#coq:tacn.congruence. Accessed 2020-04-20.

[2] Pierre Corbineau. Deciding equality in the constructor theory. In *International Workshop on Types for Proofs and Programs*, pages 78–92. Springer, 2006.

[3] Peter J Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM (JACM)*, 27(4):758–771, 1980.

[4] Robert Nieuwenhuis and Albert Oliveras. Proof-producing congruence closure. In *International Conference on Rewriting Techniques and Applications*, pages 453–468. Springer, 2005.

[5] Daniel Selsam and Leonardo de Moura. Congruence closure in intensional type theory. In *Automated Reasoning, IJCAR 2016*. Springer, 2016.