# À la Nelson-Oppen Combination for `congruence`, `lia` and `lra`

Frédéric Besson

Inria, Rennes, France

**Abstract**

We propose a tactic for combining decision procedures using a black-box Nelson-Oppen scheme. The tactic is instantiated for `congruence` and either `lia` or `lra`. The development is available at https://gitlab.inria.fr/fbesson/itauto.

## 1 Introduction

The Coq proof-assistant provides decision procedures for various logic fragments. In practice, most of the goals do not fall in those restricted fragments and, in that case, an interactive proof is required. However, there is sometimes a sweet spot when the goal can be solved by a collaboration of decision procedures. For instance, `intuition tac` enhances the expressive power of a tactic `tac` by providing support for propositional logic. Our recent `itauto tac` [3] shares the same goal but aims at improving the completeness and efficiency of the combination.

Unfortunately, there is currently no support for solving goals that are expressed in the combined decidable logic fragments of EUF [1] (Equality Logic with Uninterpreted Functions) and LIA [7] (Linear Integer Arithmetic). Yet, `congruence`[1] [5] subsumes EUF and `lia`[2] [4, 2] solves LIA. Moreover, Nelson and Oppen [6] propose a combination scheme which is complete for the combination EUF+LIA.

In the following, we present our `smt` tactic[3] which implements the Nelson-Oppen combination scheme in a black-box manner.

## 2 Motivating Example

The crux of the Nelson-Oppen scheme is that equality sharing is sufficient[4] for a complete combination of two decidable theories when the unique shared symbol is equality. The following example illustrates a somewhat painful interactive proof that is automated by our `smt` tactic.

**Example 1.** *Consider the following goal.*

```
Goal ∀ (x y: Z) (P:Z -> Prop), x :: nil = y + 1 :: nil -> P (x - y) -> P 1.
```

*Neither* `congruence` *nor* `lia` *solves the goal. Yet, it can be solved by only asserting equalities that are solved by either* `congruence` *or* `lia`*. This is illustrated by the following proof script.*

```
Proof. intros. assert (x = y+1) by congruence. assert (x-y = 1) by lia. congruence. Qed.
```

## 3 Nelson-Oppen Algorithm

The first task of the `smt` tactic is the so-called *purification* phase which identifies terms that are shared across theories. The second task consists in propagating equalities between *pure* terms until the goal is solved. These two phases are implemented as an OCaml plugin.

[1] https://coq.inria.fr/distrib/V8.13.0/refman/proofs/automatic-tactics/logic.html#coq:tacn.congruence
[2] https://coq.inria.fr/distrib/V8.13.0/refman/addendum/micromega.html#coq:tacn.lia
[3] https://gitlab.inria.fr/fbesson/itauto
[4] Under technical conditions that are not detailed here

**Purification** The *purification* introduces fresh variables and equations so that every term belongs to one and only one theory. For Example 1, we would obtain the following goal.

```
hpr1 : 1 = pr1,  hpr3 : y + pr1 = pr3,  hpr2 : x - y = pr2
H  : x :: nil = pr3 :: nil,  H0 : P pr2
============================================================
P pr1
```

The set of potential equations is then defined as $\{x = y \mid (x, y) \in Var \times Var\}$ where $Var = \{\texttt{pr1}, \texttt{pr2}, \texttt{pr3}, \texttt{x}\}$. The set of variables contains fresh variables but also the existing variables that are at the interface between the two theories. Here, the variable `x` is an arithmetic variable using as argument of the constructor `::`.

**Theory Description** In order to perform the purification phase, it is necessary to have the signature of the theory that is combined with EUF *i.e.*, the set of arithmetic types and operators. This is done by declaring instances of the two following type-classes.

```
Class TheoryType(Tid:Type)(T:Type):Type. Class TheorySig(Tid:Type){T:Type}(Op:T):Type.
```

Note that the type-classes are parametrised by an uninterpreted type `Tid` that only used to identify a theory. In our case, `ZarithThy` is associated to `lia` and `RarithThy` is associated to `lra`.

**Equality Sharing** Our current Nelson-Oppen tactic is binary and can combine `congruence` with either `lia` or `lra`. After purification, we recursively try to prove one of the equality using either `congruence` or the arithmetic tactic. If tactic $T_1$ succeeds at asserting an equation, we try to solve the goal using tactic $T_2$. If $T_2$ fails, we iterate the process until none of the equation can be proved. As there is a quadratic number of possible equations, the combination requires, in the worst case, a cubic number of calls to the decision procedures.

# 4 Conclusion and Limitations

Our `smt` tactic improves automation and has the advantage of reusing the existing tactics `congruence`, `lia` and `lra` in a black-box manner. More experiments are needed to assess to what extent automation is increased in practice and whether efficiency is satisfactory. Improving efficiency would require to adapt the decision procedures so that they either prove a goal or assert equations. Though this might not be a problem in practice, our implementation is not complete. LIA is a so-called *non-convex* theory which requires propagating not only equalities but disjunctions of equalities. Moreover, `lia` is first running `zify`. It is currently unclear how this impacts the Nelson-Oppen scheme.

# References


[1] W. Ackermann. *Solvable cases of the decision problem.* Studies in logic and the foundations of mathematics. North-Holland Pub. Co., Amsterdam, 1954.

[2] F. Besson. Fast reflexive arithmetic tactics the linear case and beyond. In *TYPES*, volume 4502 of *LNCS*, pages 48–62. Springer, 2006.

[3] F. Besson. Itauto: an Extensible Intuitionistic SAT Solver. In *ITP*, volume 193 of *LIPICS*. Dagstuhl, 2021.

[4] F. Besson, P.-E. Cornilleau, and D. Pichardie. Modular SMT Proofs for Fast Reflexive Checking Inside Coq. In *CPP*, volume 7086 of *LNCS*, pages 151–166. Springer, 2011.

[5] P. Corbineau. Deciding Equality in the Constructor Theory. In *TYPES*, volume 4502 of *LNCS*, pages 78–92. Springer, 2006.

[6] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[7] M. Presburger. *Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt.* 1931.