Transferring Arithmetic Decision Procedures (on Z) to Alternative Representations

Nicolas Magaud

Transferring Arithmetic Decision Procedures (on Z) to Alternative Representations

Nicolas Magaud
Icube UMR 7357 CNRS - Université de Strasbourg (IGG) & INRIA Nancy Grand-Est (Camus)
magaud@unistra.fr

Abstract

We propose a generic approach to make arithmetic decision procedures designed for the concrete data-type `Z` of `Coq` available for alternative representations of integers. It is based on a `transfer` tool recently developped by Zimmermann and Herbelin to perform automatic and transparent transfer of theorems along isomorphisms.

1. Introduction

Mathematical concepts have several implementations in computer science. All these different representations of the same objects can be abstracted away into a single interface, e.g. by using modules in Coq. However, decision procedures usually rely on a particular implementation and can not be easily reused when considering an alternative representation.

Coq (Coq development team 2016; Bertot and Castéran 2004) and its extension ssreflect/Mathematical Components (Gonthier et al. 2015; Mahboubi and Tassi 2016) feature at least three different implementations of the ordered set of integers and its operations ($\mathbb{Z}$,+,*,<), namely `Z` , `bigZ` and `int`.

The historical representation of integers in Coq, namely `Z` comes together with several arithmetic decision procedures including `omega`, `lia` and `psatz` (Besson 2006). While tactics such as `ring` can be (re-)instantiated easily (Grégoire and Mahboubi 2005), the above-mentionned decision procedures are designed to be used only on the representation `Z` of integers. One can not use them to prove statements featuring objects of type `int` or `bigZ`.

In this abstract, we present an approach based on some recent work (Zimmermann and Herbelin 2015) to transfer theorems from one setting to another. We use this infrastructure to transfer statements dealing with objects of type `int` into statements on objects of type `Z` and then conclude by applying the appropriate decision procedure on the transferred statement.

2. Changing Data Representation

The plugin `transfer` of Zimmermann and Herbelin automatically transfer theorems from one representation to another. It proceeeds in a transparent way for the programmer/prover. It has been successfully used to transform statements on binary natural numbers `N` into unary natural numbers `nat`.

CoqPL'17 January, 2017, Paris, France

ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00
DOI: http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

In practice, it features a tactic called `transfer` which performs the transformation following information provided by the user as instances of the class `Related`. We shall thus define a relation `R` between elements of `int` and `Z` and show that it is an instance of the class `Related`. Then we shall prove the relation `R` is total and bijective as well as some compatibility properties with respect to the usual operations on integers.

2.1 Data Representations

On the one hand, the data-type `Z` of Coq (defined in the library `ZArith`) is an inductive data-structure with three constructors (`Pos` n, `Neg` n and `Z0`) where n is a non-zero natural number (using binary representation). On the other hand, the data-type `int` of the library Mathematical Components is built as a mirror type duplicating the type `nat` (with an offset) for negative numbers. It has two constructors (`Posz` n and `Negz` n where n is an (unary) natural number). Operations and predicates are defined following the structure of each data-type. Therefore reasoning mechanisms (and decision procedures) designed for `Z` can not be easily re-used in the other setting.

2.2 Requirements

We define two functions f : `int` $\rightarrow$ `Z` and g : `Z` $\rightarrow$ `int` which show how to go from one implementation to the other. Then we define a new relation `R` which connects equivalent objects in the two types `int` and `Z`. This relation will be the one the `transfer` plugin uses to figure out which transformations to perform.

```
Definition f (x: int) := match x with
    Posz t => Z.of_nat t
  | Negz t => (- (Z.of_nat t)) - (1%Z)
  end.

Definition g (x:Z) : int := match x with
    Z0 => Posz 0%nat
  | Zpos p => Posz (nat_of_pos p)
  | Zneg p => Negz ((nat_of_pos p).-1)
  end.

Definition R (a:Z) (b:int) := a = f b.
```

We show that the relation R has the expected properties w.r.t. to the class `Related`, which boils down to checking that both $\forall x$: `int`, `R` $(f\ x)\ x$ and $\forall x$: `Z`, `R` $x\ (g\ x)$ hold.

```
Instance rel1 : forall x:int, Related R (f x) x | 10.
Instance rel2 : forall x:Z, Related R x (g x) | 10.
```

We also prove that the relation `R` is functional and injective as well as total and surjective. In Coq, these properties are written using the respectful arrow `##>`. The statement `biunique_R` expresses injectivity (right-uniqueness) and functionality (left-uniqueness)

whereas the statement `bitotal_R` expresses surjectivity (right-totality) and (left-)totality.

```
Instance biunique_R :
  Related (R ##> R ##> iff) (@eq Z) (@eq int).

Instance bitotal_R :
  Related ((R ##> iff) ##> iff) (@all Z) (@all int).
```

The next steps consist in stating that standard operations/predicates on integers are compatible with the relation `R`. We only give the statements for addition and the order *less or equal*. Note that a technical issue forces us (for the time being) to use additionnal dummy definitions such as `ler'` which is a simple coercion of the relation `ler` into `Prop` so that the transfer infrastructure recognizes the predicates which need to be transferred.

```
Instance add_transfer :
  Related (R ##> R ##> R) Z.add intZmod.addz.

Definition ler' : int -> int -> Prop := Num.Def.ler.
Instance le_transfer :
  Related (R ##> R ##> iff) Z.le ler'.
```

Once the correspondence is well established, we can do proofs and use our transfer properties automatically through extensions of the `transfer` tactic.

2.3 Tactics

Provided decision procedures are called `tomega`, `tlia`, `tpsatz`. They simply invoke the `transfer` tactic followed by the appropriate decision procedure, thus proving the goal in one line.

2.4 Dealing with the small-scale reflection feature

Predicates defined in the library MathComp are functions from `int` into `bool` whereas predicates of `ZArith` are functions from `Z` to `Prop`. However the (invisible) coercion from `bool` to `Prop` allows for a smooth connection between these predicates.

3. Results

Once we have proved in Coq the relation `R` is an instance of the class `Related`, we benchmark the system using the test suite of `micromega`[1]. We assume all the statements of these files were made on objects of type `int` and successfully translate them into equivalent statements on the corresponding objects of type `Z`. This includes proving with `tlia` the *omega nightmare* (Pugh 1991) which states :

$$\forall x\ y : \texttt{int}, 27 \leq 11{*}x{+}13{*}y \leq 45 \rightarrow \neg(-10 \leq 7{*}x{-}9{*}y \leq 4)$$

Using the datatype `Z`, `lia` requires 0.019 s to solve the goal whereas using the datatype `int`, `tlia` requires 0.43 s. The overhead corresponds to performing instance resolution to transfer the goal (using `int`'s) into the same statement using `Z`'s.

The only remaining challenge is the so-called *motivational example* - as it is named in the `micromega` library - which is not a first-order formula and therefore is not (yet) handled by the `transfer` tool of Zimmermann and Herbelin.

The implementation as well as the examples can be downloaded from `http://dpt-info.u-strasbg.fr/~magaud/SSR`.

4. Alternative approaches and related work

Our initial approach was to make the implementation of the decision procedures dealing with arithmetic independent of the representation of integers. However most of the implementation is in Ocaml and intricately tied to the actual representation of `Z` which is mapped to the `Ocaml` data-type `bigint`. It appeared easier to use the `transfer` plugin instead.

Mahboubi and Sibut-Pinote also proposed an *ad-hoc* preprocessor which transform goals featuring Mathematical Components style operations to make them fit to be processed by the `lia`/`omega` tactics (Chyzak et al. 2014).

5. Conclusions and future work

We propose an elegant and generic way to reuse decision procedures designed on a particular data-type with an alternative data-type. This is applied successfully to statements in `int` which can be proved using transparent transfer and then the appropriate arithmetic decision procedure in `Z`. The proposed infrastructure has several advantages. First it does not require any writing of ML code. Second, it can be reused for any new decision procedure provided in `Z`. Indeed, it works independently of the actual code of the decision procedure.

Work in progress includes dealing with other data-types, e.g `bigZ`. We also investigate how to make such tactics usable in the context of non-standard arithmetic. In (Magaud et al. 2014), we build a non-standard representation of the real line, based on sequences of integers (Laugwitz-Schmieden integers). A subset of these sequences of integers (the standard ones) correspond to actual integers and thus decision procedures such as `omega` could be used.

Acknowledgments

The author wishes to thank Théo Zimmermann for helping him understand the behavior of the `transfer` tactic.

References

Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

F. Besson. Fast Reflexive Arithmetic Tactics the Linear Case and Beyond. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs TYPES*, volume 4502 of *LNCS*, pages 48–62. Springer, 2006.

F. Chyzak, A. Mahboubi, T. Sibut-Pinote, and E. Tassi. A Computer-Algebra-Based Formal Proof of the Irrationality of $\zeta(3)$. In *ITP - 5th International Conference on Interactive Theorem Proving*, Vienna, Austria, 2014. URL `https://hal.inria.fr/hal-00984057`.

Coq development team. *The Coq Proof Assistant Reference Manual, Version 8.5*. LogiCal Project, 2016. URL `http://coq.inria.fr`.

G. Gonthier, A. Mahboubi, and E. Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, Inria Saclay Ile de France, 2015. URL `https://hal.inria.fr/inria-00258384`.

B. Grégoire and A. Mahboubi. Proving Equalities in a Commutative Ring Done Right in Coq. In J. Hurd and T. F. Melham, editors, *Theorem Proving in Higher Order Logics (TPHOLs), Proceedings*, volume 3603 of *LNCS*, pages 98–113. Springer, 2005.

N. Magaud, A. Chollet, and L. Fuchs. Formalizing a Discrete Model of the Continuum in Coq from a Discrete Geometry Perspective. *Annals of Mathematics and Artificial Intelligence*, 74(3-4):309–332, 2014. URL `https://hal.inria.fr/hal-01066671`.

A. Mahboubi and E. Tassi. *Mathematical Components*. Draft, 2016. URL `https://math-comp.github.io/mcb/`.

W. Pugh. The Omega Test: a Fast and Practical Integer Programming Algorithm for Dependence Analysis. In *Proceedings Supercomputing '91*, pages 4–13. ACM, 1991.

T. Zimmermann and H. Herbelin. Automatic and Transparent Transfer of Theorems along Isomorphisms in the Coq Proof Assistant. Conference on Intelligent Computer Mathematics (CICM 2015), May 2015. URL `https://hal.archives-ouvertes.fr/hal-01152588`.

[1] namely the files `zomicron.v`, `example.v` and `bertot.v`.