Transferring lemmas and proofs in Isabelle/HOL: a survey[1]

Jesús Aransay

UNIVERSIDAD DE LA RIOJA

Seminario de Informática Mirian Andrés
11th October 2016

[1] This work has been supported by the project MTM2014-54151-P from Ministerio de Economía y Competitividad (Gobierno de España)

Goal

Situations arise (in ITP) where different representations of data types are (needed?) used

You shall not be blamed for this, but you have to cope with it...

Goal

Situations arise (in ITP) where different representations of data types and versions of algorithms are (needed?) used

For instance *vectors* (and therefore, matrix representation)

- dense (lists, arrays, etc.)
- sparse (lists, arrays, etc.)
- functions (over the naturals, with a finite number of indexes with nonzero value)
- pairs of a natural number and a function from the naturals to the vector elements
- functions (over a finite type of indexes)
- etc.

Goal

Situations arise (in ITP) where different representations of data types and algorithms are (needed?) used

Archive of Formal Proofs

As a matter of example, in the Isabelle Archive of Formal Proofs, at least 10 entries directly mention the word “Refinement”, in combination with:

- Framework
- Imperative Programs
- Calculus
- Data
- Monadic Programs
- etc.

Goal

Purpose

We illustrate some examples and standard methodologies in Isabelle/HOL to (manually or automatically) transfer propositions among data type representations

Sorted from more to less “universally” applicable

- Manually: code generation
- Automatically: Lift and Transfer
- Automatically: Types to sets
- Automatically: Isabelle Refinement Framework

I will present them in reverse order

Disclaimer

Any omissions and mistakes in this survey are my own and only responsibility

Isabelle Refinement Framework

Author

Peter Lammich

Goal

To verify graph and automata algorithms, starting from *abstract algorithms* and refining them to *concrete implementations*

Components

- Isabelle Collections Framework
- Refinement for monadic programs
- Automatic Data Refinement
- Imperative Refinement Framework

Isabelle Refinement Framework

Isabelle Collections Framework (ICF)

ICF

- It provides uniform interfaces to various "abstract" data structures (sets, maps, sequences). *Locales* are used to represent Java interfaces
- It contains concrete implementations of such structures (by means of red-black trees, lists, hashing, tries): *data refinement*
- It offers heuristics that select, in the code generation phase, the best concrete structure for a given interface

P. Lammich. Collections Framework. Archive of Formal Proofs. `http://isa-afp.org/entries/Collections.shtml`. 2008

Isabelle Refinement Framework

Refinement for monadic programs

Monadic programs

- Based on the idea of *stepwise refinement* (both of algorithms and data types)
- It introduces *relations* (instead of functions) and relational programming to support *nondeterminism* ($\epsilon x.\ P\ x$)
- Programs are represented by a *nondeterminism monad* on which a refinement calculus is defined
- There is no support for state representation

P. Lammich. Refinement for Monadic Programs. Archive of Formal Proofs. http://isa-afp.org/entries/Refine_Monadic.shtml. 2012

Isabelle Refinement Framework

Automatic Data Refinement

Autoref

- Automatically refines algorithms over abstract concepts to algorithms over concrete implementations
- It uses *relational parametricity* in data type refinements
- It improves the degree of automation of the ICF and the IRF

P. Lammich. Automatic Data Refinement. Archive of Formal Proofs.
https://www.isa-afp.org/entries/Automatic_Refinement.shtml. 2013

Isabelle Refinement Framework

Automatic Data Refinement

Autoref - Related work

- In our proof of the echelon form, a similar idea is implemented *ad hoc* to prove the existence of a reduced row echelon form in *Bezout domains* and its computability in *Euclidean domains*
- The algorithm `echelon-form-of` is parameterised by a function `bezout`
- The correctness of the algorithm is proved in Bézout domains subject to the existence of the `bezout` function
- This premise holds in Euclidean domains (thus, it is removed)

J. Aransay, J. Divasón. Formalisation of the Computation of the Echelon Form of a Matrix in Isabelle/HOL. Formal Aspects of Computing. 2016.

Isabelle Refinement Framework

Imperative Refinement Framework

- It is based on Imperative/HOL, where a *heap exception* monad is used to represent *imperative programs*
- Abstract programs in the *nondeterminism monad* of IRF are refined to the *heap exception* monad
- The Imperative/HOL program and the *refinement proof* are automatically synthesised (apart from hints about which imperative data structures to use)

P. Lammich. The Imperative Refinement Framework. Archive of Formal Proofs. `https://www.isa-afp.org/entries/Refine_Imperative_HOL.shtml`. 2016

Transfer and Lifting

Authors

Brian Huffman and Ondřej Kunčar

Defining new types

- Two common ways to define new types in Isabelle/HOL are:
  - as *quotient types* ($\mathbb{Z}$ from $\mathbb{N}$, $\mathbb{Q}$ from $\mathbb{Z}$)
  - as *subsets* of existing types (`fset` as a subset of `set`)
- *Lifting* is a utility which allows users to define constants in these new types from existing constants in the original types
- *Transfer* is a tool that enables to automatically transfer propositions between two different types

B. Huffman, and O. Kunčar Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. CPP 2013. pp. 131 – 146. 2016

Transfer and Lifting

Transfer

- Based on the idea of *relational parametricity* (Reynolds, Wadler)
- A relation between two types (one “abstract” and one “raw”) is defined
- Then, relators among (some) constants and functions of each type are defined
- The Transfer package automatically proves the equivalence of propositions in the “raw” and “abstract” types, and permits to prove any of the versions

Transfer and Lifting

Transfer

Example - Obtained from the Isabelle Library (FSet.thy, Kunčar, Kaliszyk, Urban and Popescu)

- *Finite sets* are defined as a type *fset* (the set of sets which are finite)
- A relation is defined among the types *fset* and *set* (by means of *setup-lifting*)
- A relator is defined among the operation *finsert* for finite sets and *insert* of sets (by means of *lift-definition*)
- The proposition *finsert-commute*, proven for type *set*, is automatically proved for *fset* (see next slide)

Transfer and Lifting

Transfer

typedef $'a$ fset = {A :: $'a$ set. finite A} **morphisms** fset Abs-fset ...

setup-lifting type-definition-fset

lift-definition finsert :: $'a \Rightarrow 'a$ fset $\Rightarrow 'a$ fset **is** insert ...

lemmas finsert-commute = insert-commute [Transfer.transferred]

lemma insert-commute: insert x (insert y (A::α set)) = insert y (insert x A)
lemma finsert-commute: finsert x (finsert y (A::α fset)) = finsert y (finsert x A)

Transfer and Lifting

Lifting

- As already seen in the previous slide, *Lifting* allows to lift terms from the raw (underlying) to the abstract (new) type
- The package supports four kind of abstraction types: type copies, subtypes, total quotients, and partial quotients
- The command `setup-lifting` together with the new type definition $(Rep, Abs, \{x.\ P\ x\})$, proves that the abstract type is a quotient of the raw type

Transfer and Lifting

Lifting

- The command `lift-definition` imposes a *respectfulness* proof obligation (for instance, in the case of `finsert`)

  lemma finite s $\Longrightarrow$ finite (insert a s)

- Once the theorem has been proved, the package defines the new constant and the transfer rule

Transfer and Lifting

Remarkable examples

- Types *int*, *rat*, and *real* in the Isabelle distribution
- Types *fset* (from *set* and as a quotient of *list*), red-black trees (as a subset of trees)
- The lifting of (the type of) vectors in HOL Analysis Library (functions with a finite domain) and the Jordan Normal Form library (pairs of dimension and a characteristic function, and already a Lifting type) allowed to transfer Brouwer's fixpoint theorem from HA to JNF

J. Divasón, O. Kunčar, R. Thiemann, and A. Yamada. Perron-Frobenius Theorem for Spectral Radius Analysis. Archive of Formal Proofs. `http://isa-afp.org/entries/Perron_Frobenius.shtml` 2016

From types to sets

Authors

Ondřej Kunčar and Andrei Popescu

Local typedef

- Several concepts are defined and theorems are proved in Isabelle/HOL over types
- Working over types is cleaner
- Sometimes *sets* are preferred; they permit to use subdomains
- *Local typedef* is a mechanism to locally define types which are isomorphic to sets
- Properties are proved over this type, and transferred to the original set
- The *Transfer* tool can be applied to prove that the theorem over a type also holds over its isomorphic set

From types to sets

Local typedef - Remarkable case studies

- Topological proofs: "every compact set is closed"
- Berlekamp's Factorisation algorithm; an algorithm over records and sets is defined, and *formalised* using its *types* version

O. Kunčar, A. Popescu. From Types to Sets by Local Type Definitions in Higher-Order Logic. Interactive Theorem Proving 2016

J. Divasón, S. Joosten, R. Thiemann, and A. Yamada. A formalization of the Berlekamp-Zassenhauss Factorization algorithm. Draft. 2016

Code generation

Authors

Florian Haftmann

- The previous tools may be not enough to *communicate* any two representations
- For instance, types may be unrelated (no `Lifting` possible)
- There is always the possibility of developing an *ad hoc* connection between both representations by means of code lemmas
- This representation lives entirely in the logical level, but is completely done "by hand"

Code generation

Examples

The connection between the raw type of *abstract vectors* (as the set of every function with finite domain) and

- the type of every function with finite domain
- the type of immutable arrays

J. Aransay, J. Divasón. Formalisation in higher-order logic and code generation to functional languages of the Gauss-Jordan algorithm. Journal of Functional Programming, 2015

Some use data

P. Lammich. Collections Framework. Archive of Formal Proofs. http://isa-afp.org/entries/Collections.shtml. 2008
Used by 14 AFP entries

P. Lammich. Refinement for Monadic Programs. Archive of Formal Proofs. http://isa-afp.org/entries/Refine_Monadic.shtml. 2012
Used by 4 AFP entries

P. Lammich. Automatic Data Refinement. Archive of Formal Proofs. https://www.isa-afp.org/entries/Automatic_Refinement.shtml. 2013
Used by 11 AFP entries

P. Lammich. The Imperative Refinement Framework. Archive of Formal Proofs. https://www.isa-afp.org/entries/Refine_Imperative_HOL.shtml. 2016
Used by 1 AFP entries

Some use data

- *Lifting and Transfer*; its use is ubiquitous along the Isabellle/HOL Library
- *Code generation*; its use is ubiquitous along the Isabellle/HOL Library

Some other works on Refinements

- D. Cock, G. Klein, and T. Sewell. Secure Microkernels, state monad and scalable refinement. TPHOLs'08. pp. 167 – 182. 2008
- T. Murray, R. Sison, E. Pierzchalski and Christine Rizkallah. Compositional Security-Preserving Refinement for Concurrent Imperative Programs. Archive of Formal Proofs. https://www.isa-afp.org/entries/Dependent_SIFUM_Refinement.shtml. 2016
  Used by 0 AFP entries
- V. Preoteasa. Formalization of Refinement Calculus for Reactive Systems. Archive of Formal Proofs. https://www.isa-afp.org/entries/RefinementReactive.shtml. 2014
  Used by 0 AFP entries
- A. Coglio. Pop-Refinement. Archive of Formal Proofs. https://www.isa-afp.org/entries/Pop_Refinement.shtml. 2014
  Used by 0 AFP entries

Some other works on Refinements

- A. Armstrong, V. B. F. Gomes and G. Struth. Kleene Algebra with Tests and Demonic Refinement Algebras. Archive of Formal Proofs. `http://isa-afp.org/entries/KAT_and_DRA.shtml`. 2014
  Used by 1 AFP entries
- V. Preoteasa and R-J. Back. Verification of the Deutsch-Schorr-Waite Graph Marking Algorithm using Data Refinement. Archive of Formal Proofs. `http://isa-afp.org/entries/GraphMarkingIBP.shtml`. 2010
  Used by 0 AFP entries
- V. Preoteasa and R-J. Back. Semantics and Data Refinement of Invariant Based Programs. Archive of Formal Proofs. `http://isa-afp.org/entries/DataRefinementIBP.shtml`. 2010
  Used by 1 AFP entries
- K. Zee and V. Kuncak. File Refinement. Archive of Formal Proofs. `http://isa-afp.org/entries/FileRefinement.shtml`. 2004. Used by 0 AFP entries

Conclusions

- Transferring proofs among different representations of data types is relevant
- Transferring proofs among different versions of algorithms is relevant
- Particular solutions are developed to solve corner cases
- Standard solutions and designs are offered and heavily used in the Isabelle distribution (be sure not to reinvent the wheel!)

thank you!