Reusing Formal Proofs Through Isomorphisms*

Invited Talk

Mauricio Ayala-Rincón†
Departments of Mathematics and Computer Science
Universidade de Brasília
Brasília D.F., Brazil
ayala@unb.br

ABSTRACT

Formalization of computational objects, software and hardware, is the unique manner to guarantee well-behavior of computer programs and hardware, at least from the mathematical and logical point of view. Several verification and testing approaches have been proved of great applicability in this area being their usability made evident through real applications in the development of critical systems. Reusing the given correctness proofs of a specification in order to verify that an improved version of the originally given specification is also correct requires a great deal of effort and in several cases simple algorithmical improvements make it necessary the development of new correctness proofs from scratch. This paper sketches a methodology based on construction of isomorphisms between data structures that allows reusing correctness proofs for specifications that are obtained basically changing data structures. The case of study is a specification of the well-known Dolev-Yao cryptographic model in which the characterization of security was formalized in the proof assistant PVS. The given formalized specification was based on the representation of sequences of cryptographic operators via a data structure of finite sequences. The "improvement" consists of a specification in which lists will be used instead finite sequences.

Categories and Subject Descriptors

F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Specification techniques; F.4.1 [**Mathematical Logic**]: Proof theory; D.2.4 [**Software/Program Verification**]: Formal methods

Keywords

Specification and verification, Formalization, Proof assistants - PVS, Reusing formal proofs

*Research funded by the Brazilian Research Council CNPq and the District Federal Foundation for Research FAPDF.

†Author partially funded by the Brazilian Research Council CNPq.

1. INTRODUCTION

Formal methods are of great usability to certify quality of software and hardware design, but reusing mechanical demonstrations after the original design is modified, usually requires rebuilding proofs from scratch.

As an example, consider the following specification of a searching function of keys over lists of naturals written in the language of the well-known proof assistant PVS[1] [3].

```
search(i : nat, l : list[nat]) : RECURSIVE nat =
  IF null?(l) THEN length(l)
  ELSIF car(l) = i THEN 0
     ELSE 1 + search(i, cdr(l)) ENDIF
MEASURE length(l)
```

Correctness of this searching method consists in proving that whenever given as input a natural key `i` and a list of natural keys `l`, the computed output will be

- either the length of the list, if the searched key does not occurs in the list,
- or a valid index `k` of the list, that is a natural below the length of the list, such that `i` in fact occurs at position `k` in the list `l`.

The positions of `l` are indexed from 0 to its length minus one. These correctness constraints can be stablished as the two lemmas below, respectively.

```
not_member_gives_length : LEMMA
FORALL(l : list[nat], i : nat):
  NOT member(i,l) IMPLIES search(i, l) = l`length

search_works : LEMMA
FORALL (l : list[nat], k : nat) :
  member(k, l) IMPLIES nth(l, search(k, l)) = k
```

Formalizations of both these lemmas are obtained by induction on the length of lists after working particular characteristics of the `list` abstract data type and its primitive

[1]PVS specification and verification system available at http://pvs.csl.sri.com/

operators, such as decreasingness of the length of the **cdr** of non empty lists and preservation of the contents of the list after applying **cdr**, as well as validity of the position computed expressed through a PVS *type correctness condition* (TCC):

```
search_works_TCC1: OBLIGATION
  FORALL (l: list[nat], k: nat):
    member(k, l) IMPLIES
                      search(k, l) < length[nat](l);
```

This TCC is automatically generated by the proof assistant PVS from the specification of lemma **search_works** since the function **nth** is typed as

```
nth :  [l:  list[nat], j:  below[l`length]] -> nat
```

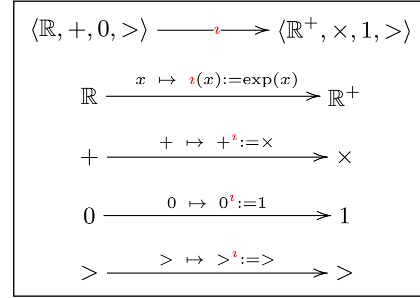
and **search_works** uses l and **search(k,l)** as arguments of **nth**.

Lists and searching on lists can be applied in several computational applications, but although all these formalizations on lists are simple, the use of lists is optional. For instance, indexation of sequences of large size requires optimal compression of information given by means of bit-code arrays and even more sophisticated data structures such as suffix and array trees. Consequently, adopting other data structure will imply the development of new specialized formalizations according to the chosen data structure of the modified specification.

Here, a formal discipline based on construction of isomorphism functors is proposed in order to reuse formalizations when different data structures are chosen to solve the same problem. The suggested approach proposes the exploration and construction of isomorphism functions together with formalization of their associated homeomorphic properties. These isomorphisms bijectively map the basic structures, relational and functional objects involved in the original given specification, for which it is supposed several formalizations were done, and the new selected data structures and their associated relational and functional objects. In this way the specifier is able to reuse formalizations previously constructed for the specification based on the original data structure.

2. BACKGROUND

Mathematically, isomorphisms are defined as a bijective transformations between algebraic structures which preserve relations and functions. Thus, an isomorphism $\imath$ maps not only elements of the domain of one structure into the other, but also functions and relations from one structure into functions and relations of the other one. For instance, *exp* is an isomorphism between $\mathbb{R}$ and $\mathbb{R}^+$, since it is a bijective function. Notice also that $exp(x+y) = exp(x) \times exp(y)$, then the corresponding operation to $+$ in $\mathbb{R}^+$ is $\times$ and vice-versa. Also, the ordering relation $>$ is preserved: $x > y$ if and only if $exp(x) > exp(y)$. Thus, the relation corresponding to $>$ in $\mathbb{R}$ is also $>$ in $\mathbb{R}^+$. Summarizing, one has a transformation $\imath$ form the structure $\langle \mathbb{R}, +, 0, > \rangle$ into the structure $\langle \mathbb{R}^+, \times, 1, > \rangle$ via *exp* in the following form:



Since *exp* is bijective, it is invertible, and one knows its inverse, denoted as $\imath$, is the function *ln*. Thus, one has two useful lemmas:

Lemma (isomorphism 1) $\imath \circ \imath$ is the identity in $\mathbb{R}$
Lemma (isomorphism 2) $\imath \circ \imath$ is the identity in $\mathbb{R}^+$

In fact, one knows that $\forall x : \mathbb{R}.\ \ln(\exp(x)) = x$ and $\forall x : \mathbb{R}^+.\ \exp(\ln(x)) = x$.

Jointly with these isomorphism lemmas, several homeomorphic properties related with the preservation of operators through the isomorphism functor are necessary:

Lemma (preservation of $+$)
$$\forall x, y : \mathbb{R}.\ \imath(x+y) = \imath(x) +^{\imath} \imath(y)$$
Lemma (preservation of > 1)
$$\forall x, y : \mathbb{R}.\ x > y \Leftrightarrow \imath(x) >^{\imath} \imath(y)$$

In fact, $\forall x, y : \mathbb{R}.\ \exp(x+y) = \exp(x) \times \exp(y)$ and $\forall x, y : \mathbb{R}.\ x > y \Leftrightarrow \exp(x) > \exp(y)$.

Also, one has homeomorphic properties related with the preservation of operators through the inverse of the isomorphic functor:

Lemma (preservation of $\times$)
$$\forall x, y : \mathbb{R}^+.\ \imath(x \times y) = \imath(x) \times^{\imath} \imath(y)$$
Lemma (preservation of > 2)
$$\forall x, y : \mathbb{R}+.\ x > y \Leftrightarrow \imath(x) >^{\imath} \imath(y)$$

In fact, $\forall x, y : \mathbb{R}^+.\ \ln(x \times y) = \ln(x) + \ln(y)$ and $\forall x, y : \mathbb{R}^+.\ x > y \Leftrightarrow \ln(x) > \ln(y)$.

Now, suppose the following equational theorems in which new operators "$-(_)$" and "$(_)^{-1}$" are involved, have been proved in $\langle \mathbb{R}, +, > \rangle$:

Theorem (additive inverse) $\forall x : \mathbb{R}.\ x + (-x) = 0$

Theorem (ln of mult. inverses) $\forall x : \mathbb{R}^+.\ \ln(x^{-1}) = -\ln(x)$

The proof of this theorem can be *reused* in order to prove new theorems in the structure $\langle \mathbb{R}^+, \times, > \rangle$, for instance

Theorem (multiplicative inverse) $\forall x : \mathbb{R}^+.\ x \times x^{-1} = 1$

can be proved as follows:

1. $x \times x^{-1} = \exp \circ \ln(x \times x^{-1})$, by Lemma isomorphism 2;

2. $\exp \circ \ln(x \times x^{-1}) = \exp(\ln(x) + \ln(x^{-1}))$, by preservation of $\times$;

3. $\exp(\ln(x) + \ln(x^{-1})) = \exp(\ln(x) + - \ln(x))$, by Theorem of ln of mult. inverses;

4. $\exp(\ln(x) + - \ln(x)) = \exp(0)$, by Theorem of additive inverse;

5. $\exp(0) = 1$, by application of the isomorphism exp.

In this way, a new proof of a theorem in the structure $\langle \mathbb{R}, +, 0, > \rangle$ is obtained from proofs in the other structure, that is $\langle \mathbb{R}^+, \times, 1, > \rangle$, by applying isomorphic properties without the need to prove additional algebraic properties in the original structure.

Of course, this kind of reuse of proofs can be applied in computational formalizations, but always depending on the specificities of the objects, functions and relations being treated.

3. ISOMORPHIC TRANSFORMATION IN COMPUTATIONAL SPECIFICATIONS

Several additional aspects need special attention when dealing with computational specifications and formalizations; among them, it deserves consideration the fact that in computation one deals with poly-sorted functions and relations. Thus, while in mathematics one deals with isomorphisms from a unique domain into another one (e.g., from $\mathbb{R}$ into $\mathbb{R}^+$) in the definition of isomorphism functor in the computational context, it is necessary to deal with transformations between a family of sorts and signatures of poly-sorted functions and relations.

A poly-sorted signature is a structure of the form $\langle \mathcal{A}, \mathcal{F}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a finite family of sorts, say $\{\tau_1, \dots, \tau_n\}$; $\mathcal{F}$ is a finite set of functions together with their types, that is, for each $f \in \mathcal{F}$, one has a type description of $f: \tau_{i_1} \times \dots \times \tau_{i_n} \to \tau$, where τ and each τ_{i_j}, for $1 \leq j \leq n$, is a type in the family of sorts; and, $\mathcal{R}$ is a finite set of predicates together with their types, that is, for each $p \in \mathcal{R}$, one has a type description of $p: \tau_{k_1} \times \dots \times \tau_{k_m}$, where each τ_{k_j}, for $1 \leq j \leq m$, is a type in the family of sorts.

This way, it is possible to define poly-sorted functions such as the element of a list of naturals, $nth : index \times List[\mathbb{N}] \to \mathbb{N}$, which is intended to take as input a valid index of a list of naturals and to give as output the natural in this position of the list.

Now, the definition of isomorphism can be stablished.

Definition (Isomorphisms between poly-sorted signatures) *Let $\langle \mathcal{A}, \mathcal{F}, \mathcal{R} \rangle$ and $\langle \mathcal{B}, \mathcal{G}, \mathcal{P} \rangle$ be signatures consisting of families of sets $\mathcal{A} = \{A_1, \dots, A_n\}$ and $\mathcal{B} = \{B_1, \dots, B_n\}$, functions $\mathcal{F} = \{f_1, \dots, f_k\}$ and $\mathcal{G} = \{g_1, \dots, g_k\}$ and relations $\mathcal{R} = \{r_1, \dots, r_l\}$ and $\mathcal{P} = \{p_1, \dots, p_l\}$. An isomorphism between these structures, $\imath$ is a bijective mapping from the families of sets, and from functions into functions and relations into relations, such that the following homeomorphic preservation properties hold:*

- *For all $f \in \mathcal{F}$, and m-tuple of well-typed arguments for f, $x_1, \dots, x_m$, supposing f is an m-ary function of type $\tau_{i_1} \times \dots \times \tau_{i_n} \to \tau$,*

$$\imath_\tau(f(x_1, \dots, x_m)) = f^\imath(\imath_{\tau_{i_1}}(x_1), \dots, \imath_{\tau_{i_m}}(x_m));$$

- *For all $p \in \mathcal{P}$, and m-tuple of well-typed arguments for p, $x_1, \dots, x_m$, supposing p is an m-ary predicate of type $\tau_{i_1} \times \dots \times \tau_{i_n}$,*

$$\imath(p(x_1, \dots, x_m)) \text{ if and only if } p^\imath(\imath_{\tau_{i_1}}(x_1), \dots, \imath_{\tau_{i_m}}(x_m)).$$

For brevity, subscripts of the isomorphic transformation are omitted, but it should be noticed that this transformation is in general polymorphic and poly-sorted. The former, since having several sorts, the equality relation is polymorphic and it has to be mapped by the isomorphism.

For our structures $\langle \mathbb{R}, +, > \rangle$, $\langle \mathbb{R}^+, \times, > \rangle$, the isomorphism $\imath$ maps $x \in \mathbb{R}$ as $\imath(x) = exp(x)$, $+^\imath$ is mapped in $\times$ and $>^\imath$ into $>$. Thus, $\imath(x+y) = \imath(x) +^\imath \imath(y)$, that is $exp(x) \times exp(y)$.

In general, isomorphisms can be sketched as in Fig. 1.

4. CASE-STUDY: LISTS VERSUS SEQUENCES IN A CRYPTOGRAPHIC FORMALIZATION

In this section, reuse of proofs through isomorphisms in a simple case of study will be considered. Alternative representation of the freely generated monoid representing chains of cryptographic operators in a formalization of the Dolev-Yao cascade protocol model ([1]) as presented in [2] and subsequently in [4] will be considered.

Essentially, a cryptographic protocol in the Dolev-Yao model is a sequence of alternating chains of cryptographic operators, which specifies a communication protocol to be obligatorily followed by the actors of a communication net. This is a two party model, but is of great relevance and usability in a great variety of current (two and multi party) protocols since it is embedded as part of most of the modern electronic protocols.

Any user $u \in U$ owns encrypt and decrypt keys E_u and D_u. The set $\{E_u \mid u \in U\} \cup \{D_u \mid u \in U\}$ is the alphabet of the language of the protocol and words freely generated by this alphabet are steps of a protocol. From the algebraic point of view, this is the freely generated monoid over this alphabet in which concatenation and empty word, denoted as λ, play the role of the binary operator and identity of the monoid. Additionally, one considers the congruences

$$E_u D_u = \lambda \qquad D_u E_u = \lambda, \quad \forall u \in U$$

Thus, for any message, or plain text M, one has $E_u(D_u(M)) = D_u(E_u(M)) = M$.

Security in this model is characterized by two properties; namely, existence of encrypt operators in the first step of

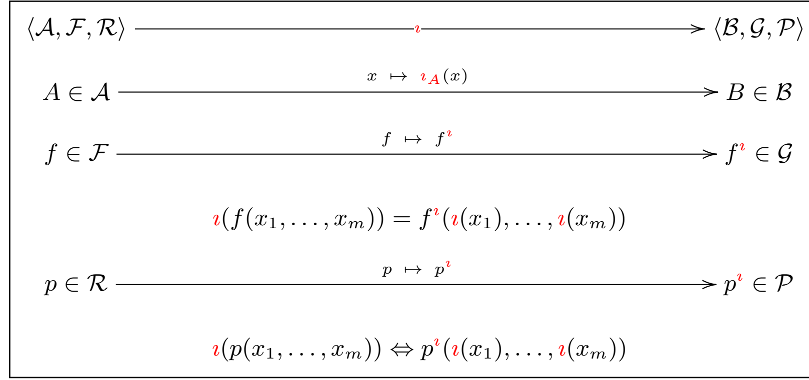


Figure 1: Isomorphism between poly-sorted signatures

the protocol and balancing property in each step of the protocol. The latter is explained as the existence of encrypt operators in any step of the protocol for any user for which a decrypt operator appears in this step of the protocol. Additionally, any malicious user, or intruder, can follow the protocol as any honest user for starting communication with any other user or continuing a communication started by another user, but also he/her can passively observe the communication between other user and eventually supplant other users. All this gives as result an admissible language for the intruder. Security or a protocol means that using this admissible language, any potential malicious user cannot extract the message hidden by the protocol. Details can be studied in the analytical proofs both in [1] and the complete PVS formalization reported in [4] for which 1.651 lines (80 KB) of specification and 55.300 lines (3.8 MB) of PVS proof commands where necessary.

The option chosen to represent monoids in [4] is the structure of finite sequences. A finite sequence is a structure of the form

```
(#length : nat, seq : [0..length-1] -> CryOp#)
```

where `length` is the length of the sequence and `seq` is the access function of the sequence which for any valid index `k`, from 0 to `length - 1`, gives as output the cryptographic operator (`CryOp`) at position `k` of the finite sequence, say `s`. This is done through calls of the form `s‘seq(k)`. By the elaborated typing discipline of the PVS specification language, whenever the type of `k` is different from `[0..length-1]`, the term `s‘seq(k)` is ill-typed.

The question of arises, from several algorithmic perspectives and different programer's point of views, whether this is the better style to specify chains of cryptographic operators. And according to the efficiency necessities (e.g., either reducing running time or space) and different programming styles, alternative data structures can be chosen instead finite sequences. For instance, instead finite sequences, ADTs as `lists` of `CryOp` can be used.

```
list[CryOp]
```

In the PVS specification language, as in other functional languages, lists of objects of type `T`, `lists[T]`, are built from the empty list, that is denoted as `null` and through the constructor `cons`, that is a function of type `T, list[T] -> T`, and for an object of type `T` and a list of type `list[T]` builds the list whose first element is the input object and whose tail is the original list. The type of lists is parameterized and PVS will automatically generate a variety of ADT lemmas including as well a correct inductive schema of proofs.

For illustration, consider reusing the proof of

Theorem(length of empty sequences)
`s‘length = 0 IFF s = empty_seq`

to prove that the following analogous result over lists.

Theorem(length of null list)
`length(l) = 0 IFF l = null`

The isomorphism functor from sequences to lists of cryptographic operators includes several transformations, as the ones presented in Fig. 2.

In order to have the isomorphic engine, all the additional inexistent necessary operators (such as $\imath$(`_‘seq`) in Fig. 2) should be specified as well as all necessary isomorphic properties should be formalized, explicitly.

In particular, on the one side, sequences are isomorphically transformed into lists through the following recursive function.

```
ı(s : seq[CryOp]) RECURSIVE : list[CryOp] =
IF s‘length = 0 THEN null
ELSE cons(s‘seq(0), ı(s(1, s‘length - 1))
ENDIF
MEASURE seq‘length
```

Additionally, several homeomorphic properties should be

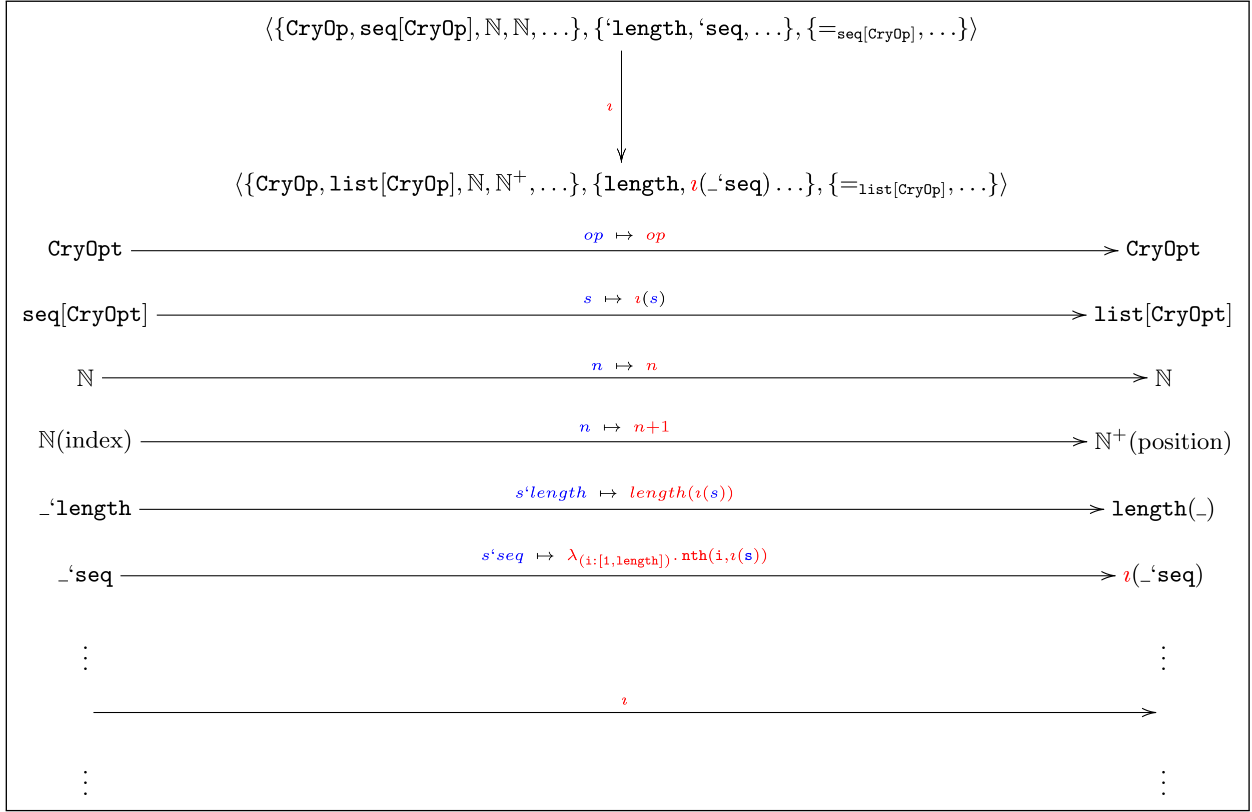


Figure 2: Isomorphism between sequences and lists of CryOps

formalized as, for instance:

Lemma A1 $\imath(\texttt{s`length}) = \texttt{length}(\imath(\texttt{s}))$
Lemma A2 $\imath(\texttt{s`seq}) = \lambda_{(\texttt{i}:[1,\texttt{s`length}])}.\texttt{nth}(\texttt{i}, \imath(\texttt{s}))$
Lemma A3 $\imath(\texttt{s`seq(k)}) = (\lambda_{(\texttt{i}:[1,\texttt{s`length}])}.\texttt{nth}(\texttt{i}, \imath(\texttt{s})))\imath(\texttt{k})$

Observe, that one has:

$(\lambda_{(\texttt{i}:[1,\texttt{s`length}])}.\texttt{nth}(\texttt{i}, \imath(\texttt{s})))\imath(\texttt{k}) \to_\beta$
$(\lambda_{(\texttt{i}:[1,\texttt{s`length}])}.\texttt{nth}(\texttt{i}, \imath(\texttt{s})))(\texttt{k}+1) \to_\beta \texttt{nth}(\texttt{k}+1, \imath(\texttt{s}))$,

thus, by lemma A3, $\imath(\texttt{s`seq(k)}) = \texttt{nth}(\texttt{k}+1, \imath(\texttt{s}))$.

On the other side, lists are isomorphically transformed into sequences through the following specified function.

```
ı(l : list[CryOp]) : seq[CryOp] =
(# length = length(l),
   seq = λ(i:[0,length(l)−1]).nth(i+1, l) #)
```

As in the direction from sequences to lists, in this direction homeomorphic properties should be formalized.

Lemma B1 $\imath(\texttt{length(l)}) = (\imath(\texttt{l}))\texttt{`length}$
Lemma B2 $\imath(\texttt{nth}(\texttt{k},\texttt{l})) = (\imath(\texttt{l}))\texttt{`seq}(\imath(\texttt{k}))$

Notice that

$\lambda_{(\texttt{i}:[0,\texttt{length(l)}-1])}.\texttt{nth}(\texttt{i}+1,\texttt{l}))(\imath(\texttt{k})) =$
$\lambda_{(\texttt{i}:[0,\texttt{length(l)}-1])}.\texttt{nth}(\texttt{i}+1,\texttt{l}))(\texttt{k}-1) \to_\beta \texttt{nth}(\texttt{k},\texttt{l})$.

A family of lemmas about isomorphic properties are necessary, among them one has:

Lemma isomorphism 1 $\forall \texttt{s} : \texttt{seq[CryOp]}.\ \imath \circ \imath(\texttt{s}) = \texttt{s}$
Lemma isomorphism 2 $\forall \texttt{l} : \texttt{list[CryOp]}.\ \imath \circ \imath(\texttt{l}) = \texttt{l}$

The previous lists of homeomorphism lemmas is not at all exhaustive, and several other isomorphic transformations should be built in order to be able to reuse proofs.

Coming back to the example of reusing the proof of Theorem `s‘length = 0 IFF s = empty_seq` to prove Theorem `length(l) = 0 IFF l = null`, one can follow the sketch below:

$\mathtt{length(l) = 0} \Leftrightarrow$ appl. of isomorphism operator
$\imath(\mathtt{length(l) = 0}) \Leftrightarrow$ isomorphism properties
$\imath(\mathtt{length(l)}) = \imath(\mathtt{0}) \Leftrightarrow$ isomorphism properties
$\imath(\mathtt{length(l)}) = \mathtt{0} \Leftrightarrow$ isomorphism properties
$\imath(\mathtt{l})$`$\mathtt{length = 0\ \ IFF}$ reuse of $\mathtt{Theorem}$
$\imath(\mathtt{l}) = \mathtt{empty_seq} \Leftrightarrow$ application of isomorphism
$\imath(\imath(\mathtt{l}) = \mathtt{empty_seq}) \Leftrightarrow$ isomorphism properties
$\imath(\imath(\mathtt{l})) = \imath(\mathtt{empty_seq}) \Leftrightarrow$ isomorphism properties
$\mathtt{l} = \imath(\mathtt{empty_seq}) \Leftrightarrow$ isomorphism properties
$\mathtt{l = null}$ □

To summarize, the approach to reuse formalizations through isomorphic transformations involves two main steps:

1. Construction and formalization of isomorphisms:
   (a) Construction of isomorphic transformations between data structures, functions and relations;
   (b) Formalization of isomorphic and homeomorphic properties;
2. Reuse of proofs.

Once the first step is completed, proofs by reusing formalizations of equational and relational theorems follow the sketches in Fig. 3 and 4, respectively.

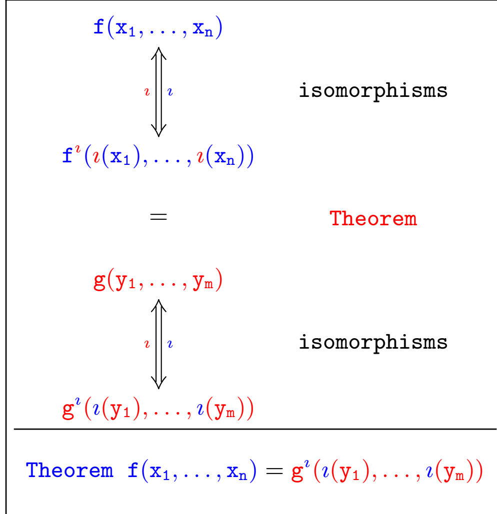


Figure 3: General sketch of reusing equational proofs by isomorphisms

- Reusing proofs is not straightforward.
- Building poly-sorted isomorphisms works well, but is an exhaustive task.
- Although this, after specifying isomorphism operators and having proved all mundane isomorphic properties complex proofs can be reused.

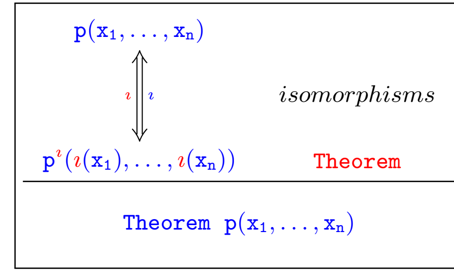


Figure 4: General sketch for reusing relational proofs by isomorphisms

5. DISCUSSIONS

Although the proposed proof reusing methodology is illustrated with very simple properties over lists and sequences, it is necessary to remark that after having developed a full PVS theory for isomorphisms between both finite sequences and lists, that includes formalizations for homeomorphic properties for functions and relations over sequences and lists, it will be possible to reuse highly elaborated formalizations of theorems of the theory of characterization of security of the Dolev-Yao model for cascade protocols. In this way, one avoids development from scratch of formalizations for the specification of the Dolev-Yao model over lists.

In general, the availability in a proof assistant of several libraries about isomorphisms between different alternative data structures is of great usability in order to adapt specifications to other data structures, different from the originally chosen, by reusing formalizations through isomorphisms. Lists and sequences are very similar, except for the inductive schemas to be adapted to deal with recursive definitions; consequently, proofs in one context are very similar to proofs in the other one. But the proposed general methodology is of great interest and usability when dealing with more elaborated data structures used for the treatment of similar solutions. For instance, elaborated data structures such as suffix trees and suffix arrays are highly explored in complex algorithmic solutions of combinatorial questions over strings, which makes of interest having isomorphic relations between them.

6. REFERENCES

[1] D. Dolev and A. C. Yao. On the Security of Public Key Protocols. *IEEE. T. on Information Theory*, 29(2):198–208, 1983.

[2] R. Nogueira, F. de Moura, A. Nascimento, and M. Ayala-Rincón. Formalization Of Security Proofs Using PVS in the Dolev-Yao Model. In *Computability in Europe CiE 2010 (Booklet)*, 2010.

[3] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11thInt. Conf. on Automated Deduction (CADE)*, volume 607 of *LNAI*, pages 748–752, 1992. Springer.

[4] Y. Rêgo and M. Ayala-Rincón. Formalization in pvs of balancing properties necessary for the security of the dolev-yao cascade protocol model. Technical Report `www.mat.unb.br/~ayala`, Universidade de Brasília, March 2012.