# Producing symmetrical facts for lists induced by the list reversal mapping in Isabelle/HOL

Martin Raška[1], Štěpán Starosta[2,*]

[1]Faculty of Mathematics and Physics, Charles University, Ke Karlovu 2027/3, 121 16 Praha 2, Czech Republic
[2]Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00 Praha 6, Czech Republic

**Abstract**
Many facts possess symmetrical counterparts that often require a separate formal proof, depending on the nature of the involved symmetry. We introduce a method in Isabelle/HOL which produces such a symmetrical fact for the list datatype and the symmetry induced by the list reversal mapping. The method is implemented as an attribute and its result is based on user-declared symmetry rules. Besides general rules, we provide rules that are aimed to be applied in the domain of Combinatorics on Words.

**Keywords**
combinatorics on words, mathematics formalizations, reversal symmetry

## 1. Introduction

While formalizing a piece of mathematical knowledge, one probably hopes that some part of the tedious work will be done by the machine. One such mechanical tasks are proofs that follow "by symmetry" which can be seen as a variation of "without loss of generality" [1]. One such "by symmetry" usually stands for a proper description of the symmetry involved and the procedure of how lemmas involving the symmetry should be used to obtain the symmetrical claim.

In this article, we exhibit a partial, yet quite useful, solution to "by symmetry" in the case of lists and the reversal mapping in the proof assistant Isabelle/HOL [2]. The reversal, or mirror mapping, is the mapping reversing the order of elements in a list. This mapping interconnects many pairs of definitions over lists in the spirit of the following duality: the list $p$ is a prefix of the list $w$ if and only if the reversal of $p$ is a suffix of the reversal of $w$. We situate this solution in the context of Combinatorics on words, a mathematical domain which studies words, i.e., lists and their various properties including equations on words.

First, we give a short overview of mathematical context along with examples of the symmetry in question. In Section 3, we shortly describe possible approaches to the solution and then we describe our solution which is part of the ongoing project of formalization of Combinatorics on Words [3]. We conclude by describing the limits of the current solution in Section 4 and conclude by final remarks in Section 5

*FMM 2021 – Fifth Workshop on Formal Mathematics for Mathematicians, 30—31 July 2021*
*Corresponding author.
stepan.starosta@fit.cvut.cz (Š. Starosta)
https://users.fit.cvut.cz/~staroste/ (Š. Starosta)
0000-0002-4414-4538 (M. Raška); 0000-0001-5962-4297 (Š. Starosta)

## 2. Mathematical context and examples of the symmetry

We work with *words*, which are finite sequences $(a_i)_{i=0}^{n}$ with $a_i \in A$ with $A$ usually being a finite set. The set of all words over $A$ is denoted $A^*$ (where $^*$ is the Kleene star). The *reversal mapping*, denoted rev, is a mapping $A^* \to A^*$ which maps the word $w = (a_i)_{i=0}^{n}$ to the word $\mathrm{rev}\,(w) = (a_{n-i})_{i=0}^{n}$, or simply put, it reads the letters of the word in the reverse order. The reversal mapping is an *involutive antimorphism* with respect to the operation of concatenation of two words, that is, $\mathrm{rev} \circ \mathrm{rev} = \mathrm{id}$ and $\mathrm{rev}\,(v \cdot w) = \mathrm{rev}\,(w) \cdot \mathrm{rev}\,(v)$ where $\cdot$ is the binary operation of concatenation. It follows that rev is also a bijection.

In our ongoing project [3] of formalization of Combinatorics on Words, we formalize many elementary preparatory lemmas dealing with a handful of notions. Many of these notions have a symmetrical counterpart, and many facts are symmetrical, and their proof is just copy and paste of the proof of the original lemma. We continue with examples that exhibit this symmetry.

### 2.1. Example 1

If a word $w$ can be written as a concatenation of two words $p$ and $s$, i.e., $w = p \cdot s$, we say that $p$ is a *prefix* of $w$ and $s$ is a *suffix*. An elementary example of a symmetrical pair of claims involving prefix and suffix is the following.

**Lemma 1.** *If $p$ is a prefix of $v$, then $p$ a prefix of $v \cdot w$.*

*Proof.* If $p$ is a prefix of $v$, there exists a word $s$ such that $v = p \cdot s$. Hence, $v \cdot w = p \cdot s \cdot w$, and $p$ is a prefix of $v \cdot w$. □

By the symmetrical counterpart of Lemma 1 we mean the following claim.

**Lemma 2.** *If $s$ is suffix of $v$, then $s$ is a suffix of $w \cdot v$.*

Its proof can be done as the presented proof of Lemma 1, however, stating in follows "by symmetry" from Lemma 1 would be no exception in literature.

In order to formally exploit the symmetry in a full proof, we have to make the intended symmetry between a prefix and a suffix explicit:

**Lemma 3.** *The word $p$ is a prefix of $w$ if and only if $\mathrm{rev}\,(p)$ is a suffix of $\mathrm{rev}\,(w)$.*

A full proof of Lemma 2, by symmetry, is as follows.

*Proof of Lemma 2.* Fix $w$ and assume that $s$ is a suffix of $v$. Let $s'$, $v'$, and $w'$ be the words such that

$$s' = \mathrm{rev}\,(s), \quad v' = \mathrm{rev}\,(v) \quad \text{and} \quad w' = \mathrm{rev}\,(w).$$

As the reversal is an involution, it follows that

$$s = \mathrm{rev}\,(s'), \quad v = \mathrm{rev}\,(v') \quad \text{and} \quad w = \mathrm{rev}\,(w').$$

As $s$ is a suffix of $v$, the word $\mathrm{rev}\,(s')$ is a suffix of $\mathrm{rev}\,(v')$. By Lemma 3, $s'$ is a prefix of $v'$. Using Lemma 1, $s'$ is a prefix of $v' \cdot w'$. Again, by Lemma 3, $\mathrm{rev}\,(s')$ is a suffix of $\mathrm{rev}\,(v' \cdot w')$. Since

$$\mathrm{rev}\,(v' \cdot w') = \mathrm{rev}\,(w') \cdot \mathrm{rev}\,(v') = w \cdot v,$$

we conclude that $s$ is a suffix of $w \cdot v$. □

## 2.2. Example 2

Since the next examples are in the framework of Isabelle/HOL, we first recall our setting. A word is represented by the datatype of list, which is is specified via 2 constructors: `Nil` (denoted []), the empty list/word, and `Cons` (denoted #), the recursive constructor allowing to add an element to the list at its beginning. The reversal mapping is represented by the function `rev`:

> **primrec** rev :: 'a list ⇒ 'a list **where**
> rev [] = [] |
> rev (x # xs) = rev xs @ [x]

with @ being the notation for list `append`, i.e., concatenation of two words.

The predicates for prefix and suffix are already part of the Isabelle distribution in the theory HOL-Library.Sublist:

> **definition** prefix :: 'a list ⇒ 'a list ⇒ bool **where** prefix xs ys ⟷ (∃zs. ys = xs @ zs)
>
> **definition** suffix :: 'a list ⇒ 'a list ⇒ bool **where** suffix xs ys = (∃zs. ys = zs @ xs)

The second example is constituted by the pair of symmetric definitions of the first and the last letter of a word, i.e., element of a list. In Isabelle/HOL, the first element of a list is its head, realized as one of two selectors, named `hd`, of the list constructor `Cons`. The last letter is the recursive function `last`:

> **primrec** (nonexhaustive) last :: 'a list ⇒ 'a **where**
> last (x # xs) = (if xs = [] then x else last xs)

given in the main theory List. To obtain a simple enough symmetry rule for `hd` and `last`, it suffices to notice that they behave the same way on the empty list.

> **lemma** hd_last_Nil: hd [] = last [] **unfolding** hd_def last_def **by** simp
>
> **lemma** hd_rev_last: hd(rev xs) = last xs **by** (induct xs, simp add: hd_last_Nil, simp)

The pair of symmetrical claims is the following.

> **lemma** example2: u ≠ [] ⟹ prefix u v ⟹ hd u = hd v
>
> **lemma** example2_sym: u ≠ [] ⟹ suffix u v ⟹ last u = last v

The goal is to obtain example2_sym from example2 by symmetry. We proceed analogously to the proof of Lemma 2 above using standard methods in Isabelle/HOL:

> example2[of rev u rev v, unfolded rev_is_Nil_conv suffix_to_prefix[symmetric] hd_rev_last]

where `rev_is_Nil_conv` is `(rev xs = []) = (xs = [])` and `suffix_to_prefix`[symmetric] is `prefix (rev xs) (rev ys) = suffix xs ys`. That is, we instantiate every variable of example2 by its reversal to obtain

> rev u ≠ [] ⟹ prefix (rev u) (rev v) ⟹ hd (rev u) = hd (rev v),

and then rewrite the terms using appropriate symmetry rules, via the unfolded attribute (which is analogous to what was done in the proof of Lemma 2 above). We end up with example2_sym and the proof by symmetry is done.

### 2.3. Example 3

The next example is the following pair of symmetric facts.

> **lemma** example3: prefix u (p @ w @ q) $\Longrightarrow$ length p $\leq$ length u $\Longrightarrow$ length u $\leq$ length (p @ w) $\Longrightarrow$
> $\exists$ r. u = p @ r $\wedge$ prefix r w
>
> **lemma** example3_sym: suffix u (p @ w @ q) $\Longrightarrow$ length q $\leq$ length u $\Longrightarrow$ length u $\leq$ length (w @ q) $\Longrightarrow$
> $\exists$ r. u = r @ q $\wedge$ suffix r w

Applying the same strategy as for example2 fails, since trying to obtain example3_sym from

> example3[of rev u rev p rev w rev q, unfolded symmetry_rules],

where `symmetry_rules` is a list of appropriate symmetry rules, leaves us with

> suffix u ((q @ w) @ p) $\Longrightarrow$ length p $\leq$ length u $\Longrightarrow$
> length u $\leq$ length (w @ p) $\Longrightarrow$ $\exists$ r. rev u = rev p @ r $\wedge$ prefix r (rev w),

which is not yet in the form of example3_sym. This is not unexpected, the claim contains a bound variable `r`. To finish the conversion, it suffices to realize that $(\exists x.\ P(x)) \leftrightarrow (\exists x.\ P(\mathrm{rev}\, x))$ holds. Thus, we may replace the last two occurrences of `r` with `rev r`, and apply appropriate symmetric rules.

The next section addresses our realization of automatic production of symmetric rules, preceded by a discussion on the use of existing tools.

## 3. Automated production of symmetrical claims

Before describing our solution to the automation of producing symmetrical claims, we discuss if and how might our task be achieved using tools for theorem reuse available in Isabelle/HOL. We have not found any ready made tool in Isabelle/HOL that could achieve our objectives. We shall briefly discuss two existing tools that achieve a similar task, namely reusing of a theory in a homomorphic setting.

The first tool are locales. Locale is a mechanism for abstraction via interpretation and locale expressions [4, 5]. We could see the "by symmetry" argument as two instantiations of the same claim: first in lists, and second in reversed lists. It would require to prove all claims about lists in an abstract setting, and then apply it to lists and reversed lists. The abstract setting would mean some kind of "axiomatic theory of lists", that is, of free monoids, as in [6]. While this may be the correct idea mathematically, we do not see how to naturally recreate it in Isabelle/HOL using locales.

The second tool is the infrastructure of transfer [7, 8]. Its main purpose is to transfer facts between two datatypes, e.g., from natural integers to integers, via user specified transfer rules. Although, in principle, it should be possible to use this powerful tool, we encountered several problems using it and we did not find a way how to employ it for our purposes without producing undesired limitations. For example, it is not clear how to specify whether in the case of transferring a fact containing `'a list list` the transfer rule should be applied to `'a list` or `'a list list = 'b list`.

Since it seems from the above discussion that there is no direct way how to achieve the desired automation of the symmetry, we propose a "lightweight" solution which closely mimics the simple reproving of each individual claim "on the fly" as indicated by the examples in Section 2. Our solution is very simple but at the same time it proves to be very practical and sufficiently versatile.

It is created as a single attribute called "reversed". The symmetry rules are collected as a list of theorems called "reversal_rule", i.e., a user can add and remove them any time. By default, rules are required to eliminate reversal images, thus the reversal images are supposed to be on the left side of the equalities serving as rules. For instance, the symmetry rule Lemma 3 is stored in this form

```
suffix (rev p) (rev w) = prefix p w.
```

The execution follows examples of Sections 2.2 and 2.3: first, all schematic variables of type list of the fact being reversed are instantiated by their reversals. Before the application of the symmetry rules, bound variables need to be treated. Let us indicate this procedure on example3 of Section 2.3 which contains one bound variable. As indicated above, the idea is to use the equivalence $(\exists x.\ P(x)) \leftrightarrow (\exists x.\ P(\text{rev}\, x))$. We introduce a helper (private) definition and 2 claims as follows:

```
definition Ex_rev_wrap :: ('a list ⇒ bool) ⇒ bool
  where Ex_rev_wrap P = (∃x. P (rev x))

lemma Ex_rev_wrapI: ∃x. P x ≡ Ex_rev_wrap P

lemma Ex_rev_wrapE: Ex_rev_wrap (λx. P x) ≡ ∃x. P (rev x)
```

The application of these claims can be seen as

```
example3[of rev u rev p rev w rev q,unfolded Ex_rev_wrapI],
```

which yields

```
prefix (rev u) (rev p @ rev w @ rev q) ⟹ length (rev p) ≤ length (rev u) ⟹
length (rev u) ≤ length (rev p @ rev w) ⟹ Ex_rev_wrap (λr. rev u = rev p @ r ∧ prefix r
(rev w)).
```

The next step is

```
example3[of rev u rev p rev w rev q,unfolded Ex_rev_wrapI, unfolded Ex_rev_wrapE]
```

resulting in

> prefix (rev u) (rev p @ rev w @ rev q) $\Longrightarrow$ length (rev p) $\leq$ length (rev u) $\Longrightarrow$
> length (rev u) $\leq$ length (rev p @ rev w) $\Longrightarrow$ $\exists$ r. rev u = rev p @ rev r $\wedge$ prefix (rev r) (rev w).

Note that the name of the bound variable is preserved in this step. It is due to ($\lambda$`x. P x`) being present in `Ex_rev_wrapE` rather than just `P`.

This two step rewriting using the definition `Ex_rev_wrap` in the intermediate step is to prevent an infinite loop of rewriting while trying to go directly from $\exists$`x. P x` to $\exists$`x. P (rev x)`.

The last form is ready for the application of symmetry rules, and we almost obtain our goal, example3_sym. The remaining difference is the order of application of @, i.e., the arrangement of parentheses. The operation @ is associative and this final adjustment is left to be done manually, if desirable.

The implementation deals with other types of bound variables in a similar manner using a definition analogous to `Ex_rev_wrap` and its two associated wrapping and unwrapping rules. In a similar spirit, a special care for the constructors `Nil` and `Cons` is also part of the reversing process. The described implementation is available at [9].

## 4. Limits of the approach

To show the current limits, consider the following pair of symmetric claims:

> **lemma** example4: prefix ps ws $\Longrightarrow$ prefix (concat ps) (concat ws)
>
> **lemma** example4_sym: suffix ps ws $\Longrightarrow$ suffix (concat ps) (concat ws)

Applying the attribute reversed on example4 produces:

> suffix ps ws $\Longrightarrow$ prefix (concat (rev ps)) (concat (rev ws))

The problem here is that we are dealing with variables of type ′`a list list`, representing factorizations or decomposition of words. As they are of type ′`b list`, the reversing happens only on this level, whereas to produce example4_sym one would need the reversing to act on ′`a list =` ′b. Namely, the additional required action of the symmetry on `ps` and `ws` is the application of `map rev`. The reason that this represents a current limit is that the choice of correct reversal rules for variables of type ′`a list list` becomes crucial and it is no more clear what are the correct reversal rules.

## 5. Concluding remarks

Although the implemented attribute seems to be very simple, together with many delicately selected reversal rules it is very useful in our current project of formalization of Combinatorics on Words [3].

As the attribute is a part of a living project, and the time period between the acceptance and publication of this article was noticeable, the obstacle exhibited in the previous section has been already surmounted in a way to suit the needs of the project. However, the goal to properly deal with variables of any type remains. In order to do that, our tentative model of the symmetry in question needs to be generalized and validated.

### Acknowledgements

The authors acknowledge support by the Czech Science Foundation grant GAČR 20-20621S.

## References

[1] J. Harrison, Without loss of generality, in: S. Berghofer, T. Nipkow, C. Urban, M. Wenzel (Eds.), Theorem Proving in Higher Order Logics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 43–59.

[2] T. Nipkow, M. Wenzel, L. C. Paulson (Eds.), Isabelle/HOL, Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45949-9.

[3] Š. Holub, Š. Starosta, et al., Combinatorics on words formalized, https://gitlab.com/formalcow/combinatorics-on-words-formalized, 2021.

[4] C. Ballarin, Tutorial to locales and locale interpretation, in: L. Lambán, A. Romero, J. Rubio (Eds.), Contribuciones Científicas en Honor de Mirian Andrés Gómez, Servicio de Publicaciones de la Universidad de La Rioja, Logroño, Spain, 2010. Also part of the Isabelle user documentation.

[5] C. Ballarin, Locales: A module system for mathematical theories, Journal of Automated Reasoning 52 (2014) 123–153. URL: https://doi.org/10.1007/s10817-013-9284-7. doi:10.1007/s10817-013-9284-7.

[6] Š. Holub, R. Veroff, Formalizing a fragment of combinatorics on words, in: J. Kari, F. Manea, I. Petre (Eds.), Unveiling Dynamics and Complexity, Springer International Publishing, Cham, 2017, pp. 24–31.

[7] B. Huffman, O. Kunčar, Lifting and transfer: A modular design for quotients in isabelle/hol, in: G. Gonthier, M. Norrish (Eds.), Certified Programs and Proofs, Springer International Publishing, Cham, 2013, pp. 131–146.

[8] O. Kunčar, Types, Abstraction and Parametric Polymorphism in Higher-Order Logic, Ph.D. thesis, Technischen Universität München, Germany, 2016.

[9] Š. Holub, M. Raška, Š. Starosta, et al., Combinatorics on words formalized: Reversal symmetry, https://gitlab.com/formalcow/combinatorics-on-words-formalized/-/tree/Archive-Reversal-Symmetry, 2021.