

Introductory Survey of Catalan Automorphisms and Bijections

Antti Karttunen

Helsinki, Finland

Email address: his-firstname.his-surname@gmail.com

Oct. 10, 2002; last revised July 26, 2011

XXX - NOTE July 2011: Although this draft is still in a very premature stage, I have now corrected some of the most glaring errors that were present in the previous Dec 2007 revision, and added proofs for a couple of cases that were previously unclear. Still, caveat lector!, many proofs might be incorrect or otherwise deficient. Also, I don't remember even myself, by which specific insight I derived certain of the formulas I give here.

For up-to-date developments on this subject, please check http://oeis.org/wiki/Catalan_bijections

XXX – TO DO: graphical illustration of general vs binary tree connection (important!), equivalence of destructive and constructive versions of recursive operations ($!ENIPS = ENIPS$, etc.), construction of A089840, "Atavism" (Proposition: Every element of A089840 occurs atavistically in ...), group structure and relationships with other groups (with Thompson's V , superficial, with $\text{Aut}(T^{(2)})$ more direct, give necessary (and conjectured to be also sufficient) condition, "psi"-homomorphism, etc. References to bijections occurring in journals.

XXX – TO DO2: Clean, clean, clean up!

Abstract

In this paper we first adopt programming language SCHEME (a dialect of LISP) as a powerful and concise way for unambiguously defining various automorphisms and arbitrarily complex bijections that act on many combinatorial structures enumerated by Catalan numbers: planar binary trees, planar general trees, parenthesizations, polygon triangulations and non-crossing chord arrangements, among others.

Then, after realizing that many of these bijections are in simple recursive relations to each other, like e.g. preorder traversal, we set out a system of "metaoperators" for recursively deriving Catalan bijections from other such bijections.

Then realizing that in turn, many of these recursive derivations can be implemented as *folds*, an important concept in Constructive Algorithmics, we apply some results from that field, and among other of its useful applications, establish inverse operators for most of our recursive operators.

We start our survey of these bijections from elementary cases, and then play with some simple methods to generate inductively infinite, countable subsets of them, which still are wide enough to encompass the most obvious symmetry operations (reflections & rotations) one can imagine to occur in the Catalan family. On the other hand we also strongly suspect that certain well-defined bijections are outside of these sets.

We summarize the formulae of the long-established OEIS-sequences which we will meet when counting the orbits and fixed points to which such automorphisms partition each subset of Catalan structures of any finite size, and derive also formulae for two new such sequences, via the orbit-counting ("Burnside's") lemma.

We also observe few specific properties these automorphisms may have, what are their implications and how various Catalan bijections are related to each other.

AMS 2000 Classification: Primary 05A15; Secondary: 05A19, 20B27, 20E08, 68P05

Contents

1 Introduction	2
2 S-expressions	6
3 A very brief introduction to the programming language Scheme	8
4 Programming reflections & rotations of common interpretations	12
5 Using destructive Scheme-primitives to implement Catalan bijections	15
6 Integer-Sequences Associated with Catalan bijections	17
7 Embeddability	21
8 Recursion Schemata	23
9 Folds	27
10 Non-recursive bijections of binary trees and their recursive derivations	36
11 Catalan bijections in detail	36
12 Relations to other groups acting on other variants of binary trees	59
13 Open questions - XXX - vague ideas	59

1. Introduction

After Fibonacci sequence, Catalan numbers has probably the most celebrated recurrence in enumerative combinatorics:

$$C_0 = 1, C_{n+1} = \sum_{i=0}^n C_i C_{n-i}$$

giving the famous terms:

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \dots$$

Because of the way the sequence is defined as a convolution of it itself, there are surprisingly many combinatorial structures which it counts. Stanley gives 66 different interpretations in his Enumerative Combinatorics, volume 2 [Stanley 1999], and several dozens more in the sequel [Stanley 2001-].

Moreover, in many cases there exists a natural isomorphism between two interpretations, which is sometimes easy to find, and sometimes not at all.

In this paper we are concerned with the following interpretations.

- (a) triangulations of a polygon with $n+2$ edges.
- (c) and (d) rooted plane binary trees of n (internal) nodes.
- (e) rooted plane general trees of n edges.
- (L) Lukasiewicz-words of those general trees.

- (i) Dyck paths of $2n$ segments. These are also called "Catalan Mountain Ranges".
- (n) noncrossing chord arrangements with n chords.
- (qq) noncrossing partitions of n .
- (rr) noncrossing Murasaki-diagrams of n stalks.
- (P) parenthesizations of n opening (closing) parentheses.

The lowercase letters refer to the sections of exercise 19 in chapter 6 of [Stanley 1999], while the uppercase letters L and P are used for those manifestations which are not explicitly present in Stanley's list. The table 1 shows each of these interpretations up to the size $n=3$.


























































n	A014486(n)	(i)	(n)	(e)	(qq)	(rr)	(c/d)	(a)
0.	0	.		.		.	—	
1.	2							
2.	10							
3.	12							
4.	42							
5.	44							
6.	50							
7.	52							
8.	56							

Table 1: The interpretations (i), (n), (e), (qq), (rr), (c/d) and (a) for the sizes 0, 1, 2 and 3 of the Catalan structures.

The natural isomorphism between the Dyck paths (i) and parenthesizations can be seen immediately, as well as between parenthesizations and plane general trees (e). Lukasiewicz-word of a

rooted plane tree is obtained by traversing it depth-first-wise from left to right, and writing down the (out-)degree of each node, leaves (terminal nodes) producing zero.

Also the interpretation (n) hides parenthesizations, which can be easily seen, when one cuts open the circular tables of noncrossing chord arrangements from the bottom, and deforms the circle to an arc. (The starting and ending vertices of each chord correspond to a pair of balanced opening and closing parentheses, see figure 1).

The interpretations (qq) and (rr) are related to each other in similar simple manner.

However, there are several natural isomorphisms between Dyck paths (i) and noncrossing Murasaki-diagrams (rr) . In this paper we will use the "right-hand side slope mapping", which operates by drawing a vertical line above each right-hand side slope, and then connecting those vertical lines that originate from the same height without any lower valleys between. This is illustrated in the figure 2, and the reader can check that it is indeed used in the table 1.

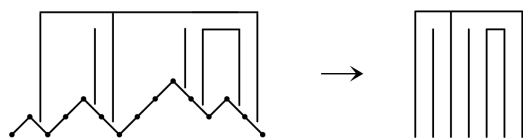
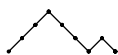


Figure 1: "Right-hand side slope mapping" defines an isomorphism between the interpretations (i) and (rr) .

At the extreme right of the figure 1 we have interpretations (c/d) and (a) . It is easy to see that each binary tree naturally defines an unique polygon triangulation, as can be seen from the figure 3 where a binary tree has been drawn inside the corresponding polygon triangulation.

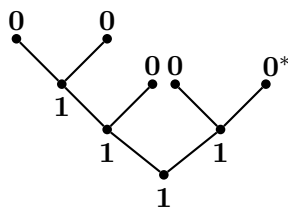
However, a natural isomorphism between (c/d) and (e) or (i) was not explicitly described until 1967 in [De Bruijn and Morselt 1967]. Between (i) and (d) it takes the following form. Proceeding from left to right, replace each upward slope $/$ of the Dyck path with 1, and downward slope \backslash with 0. For example, from



we get

11100010

from which we build a binary tree in depth-first fashion from left to right, with 1's in binary expansion standing for internal (branching) nodes, and 0's for leaves:



Note that the last leaf of the binary tree is always implicit, marked here with 0^* .

We explain the details of the direct mapping between (d) and (e) in the next section where we show how that isomorphism is implicitly present in the algorithm that converts Lisp/Scheme linked list structures from their internal to external representation.

With these isomorphisms at hand, we can fill the remaining gaps. E.g. we define an isomorphism $(a \leftrightarrow qq)$ as a composition of isomorphisms $(a \leftrightarrow d)$, $(d \leftrightarrow e)$, $(e \leftrightarrow i)$, $(i \leftrightarrow rr)$ and $(rr \leftrightarrow qq)$.

These *canonical* isomorphisms define the correspondences used in the rest of this paper. E.g. if we say that a particular Catalan automorphism rotates noncrossing partitions (qq) one step clockwise, it actually means that when the preimage and image of any binary tree (with respect to that automorphism) are transformed through such a chain of isomorphisms to noncrossing partitions, we will observe that in the interpretation (qq) the image is a clockwise rotation of the corresponding preimage.

Table 2 illustrates how a Scheme implementation of the automorphism $*A086429$, although working directly on binary trees (the interpretation (d) on the left), eventually effects the clockwise rotation of noncrossing partitions (the interpretation (qq) on the right) through the natural bijections $(d \leftrightarrow A063171)$, $(A063171 \leftrightarrow i)$, $(i \leftrightarrow rr)$ and $(rr \leftrightarrow qq)$.






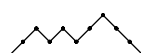



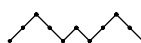



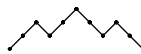






index of structure in A014486	(d)	$(A063171)$	(i)	(rr)	(qq)
29 [= A086429(39)]		1011001100			
A086429(29) = 46		1101011000			
A086429(46) = 38		1100101100			
A086429(38) = 48		1101100100			
A086429(48) = 39		1100110010			

Table 2: The effects of $*A086429$ on the interpretations (d) , $(A063171)$, (i) , (rr) and (qq) .

There are several mathematical structures that can be naturally embedded into Catalan structures, e.g. binary strings which naturally map to that subset of binary trees, which have no double-branches (we mark 0 when the binary tree branches to left, and 1 when it branches to the right, or vice versa!), as well as various Motzkin-structures that can be embedded into Catalonia by various means. Some Catalan automorphisms keep such subsets of structures closed, so they induce also an automorphism on those isomorphic structures. For example, it's easy to construct a Catalan automorphism that implements a permutation contained in binary reflected Gray Code, or one that rotates the non-crossing Motzkin-chords that are embedded in some Catalan-structure (automorphism $*A085159$ and $*A085160$). In some way such Catalan automorphisms can then be viewed as *extensions* of those automorphisms.

What's more important, because Catalan structures can be embedded into themselves, This topic is further explored in section XXX, "Embeddability".

2. S-expressions

In many cases it would be nice to have a concise and unambiguous notation for symmetry and other operations involving the structures presented in the previous chapter. Also, when examining their orbit-profiles it's useful to have some empirical data before trying to derive mathematical conclusions. Naturally, such data can be obtained with computer, by programming the rotations, reflections and other operations as procedures that act on data structures implementing various manifestations presented in Chapter I. E.g., a simple-minded approach based on the object-oriented programming (OOP) paradigm might devise a separate class for each such manifestation, with certain inheritance between the classes, e.g. a class of binary trees might be a subclass of general trees, and non-crossing handshakes and set-partitions might diverge from a common class of chord-diagrams. The objects instantiating the said classes would then each hide its own set of private elements, for whose manipulation specific methods would be needed.

Clearly this approach is far too baroque, when in the previous chapter I have already shown that these various interpretations for Catalan numbers are just different manifestations of the one and same combinatorial structure. This means that to effect, say a rotation of Euler's polygon divisions, it's enough to define a certain operation on binary trees, as long as we have a commonly agreed isomorphism from binary trees to polygon triangulations, for example the one presented in the previous chapter.

So, instead of tailoring multiple custom data types, a single universal one should suffice.

It turns out that *S-expressions*, introduced by John McCarthy in his programming language LISP [McCarthy 1960] naturally fill the bill. Being originally just designed for the manipulation of such data structures, LISP contains a powerful set of primitives for their manipulation, which have in turn been inherited by its dialect SCHEME [Sussman and Steele 1975], and to a lesser extent also by PROLOG, and certain more recent functional programming languages derived from these.

In this paper we define an *S-expression* in two different ways. Both definitions are given in BNF.

Definition 1 (S-expression in its internal, “dotted” form.)

$$\begin{aligned} S\text{-expression} &\rightarrow \text{SYMBOL} \mid \text{NUMBER} \mid () \mid \text{PAIR} \\ \text{PAIR} &\rightarrow (S\text{-expression} . \text{PAIR}) \mid (S\text{-expression} . ()) \end{aligned}$$

1

Definition 2 (*S-expression in its external, “dotless” list form.*)

$$S\text{-expression} \rightarrow \text{SYMBOL} \mid \text{NUMBER} \mid (S\text{-expression}^*)$$

Here a trailing “*” denotes zero or more occurrences. *SYMBOL* and *NUMBER* refer to symbolic and numeric terminals. `define`, `pair?`, `else` and `+` are examples of the former, and 144, -1 and 2.71828 of the latter class. “()” refers to a special terminal called either an *empty list* or *NIL*.

¹ Note that our definition of the “dotted” S-expression differs from the standard definition used in LISP and SCHEME:

$$S\text{-expression} \rightarrow \text{SYMBOL} \mid \text{NUMBER} \mid () \mid (S\text{-expression} . S\text{-expression})$$

that it doesn't allow other terminals than the empty list () on the right side of a pair.

The above definitions produce isomorphic structures. The isomorphism between the two variant syntaxes is realized by the function f which maps from the “dotted pair” to “list” syntax as follows:

$$f(s) = \begin{cases} s & \text{if } s \text{ is } \textit{NUMBER}, \textit{SYMBOL} \text{ or } () \\ (f(S\text{-expression})) & \text{if } s \text{ is of the form } (S\text{-expression} . ()) \\ \text{a list constructed by inserting} & \text{if } s \text{ is of the form } (S\text{-expression} . \textit{PAIR}) \\ f(S\text{-expression}) \text{ to the front of} & \\ \text{a list obtained with } f(\textit{PAIR}) & \end{cases}$$

Example 3. The function f maps the internal, dotted S -expression “(() . (1 . (two . (2 . ()))))” to an external, dotless list “(() 1 two 2)” and likewise, the internal, dotted S -expression “(a . ((b . ()) . ()))” to an external, dotless list “(a (b))”

If we consider a subset of S -expressions where the only allowed terminals are $()$ ’s (i.e. NIL ’s), we get the following definition:

Definition 4 (*Nihilistic S -expression in its internal, “dotted” form.*)

$$\textit{Nihilistic } S\text{-expression} \rightarrow () \mid (\textit{Nihilistic } S\text{-expression} . \textit{Nihilistic } S\text{-expression})$$

It is clear that this definition is isomorphic with rooted plane binary trees introduced in the previous chapter, one of the most familiar interpretations (c/d) of Catalan numbers. When we apply the above function f to this subset, the result is interpretation (P) (parenthesizations) surrounded by an extra pair of “(“ and “)”. We dub this interpretation as (Px) (parenthesizations surrounded by extra parentheses).

Example 5. The function f maps the internal, dotted “nihilistic” S -expressions to the interpretation (Px) as follows:

$$\begin{aligned} () & \rightarrow () \\ (() . ()) & \rightarrow (()) \\ (() . (() . ())) & \rightarrow (() ()) \\ ((() . ()) . ()) & \rightarrow ((()) \\ \textit{etc.} & \end{aligned}$$

When we discard the outermost parentheses from the produced parenthesizations, we see that the above mapping hides a map ($c/d \rightarrow i$):

$$\begin{aligned} (c/d) & & (i) \\ \cdot & \rightarrow & \cdot \\ \color{red}{\vee} & \rightarrow & \wedge \\ \color{red}{\vee} \color{red}{\vee} & \rightarrow & \wedge \wedge \\ \color{red}{\vee} \color{red}{\vee} \color{red}{\vee} & \rightarrow & \wedge \wedge \wedge \\ \textit{etc.} & & \end{aligned}$$

We leave it as an exercise to the reader to prove that the map ($c/d \rightarrow i$) produced this way is the inverse of the map ($i \rightarrow c/d$) presented in the previous chapter.

3. A very brief introduction to the programming language Scheme

All computations in SCHEME² are effected by *evaluating* function calls or so called special forms. Specifically, a program written in SCHEME is nothing more than a set of user-defined functions invoking each other in addition to built-in functions and special-forms provided by the language.

The syntax of the SCHEME is based on S-expressions defined in the previous section. Each invocation of a function (or special form) with arity n is represented as a list of length $n + 1$, where the first element of the list is the name of the called function (special form). For example, a function invocation with three arguments, represented in mathematics usually as $f(a, b, c)$ is represented in SCHEME as a list of four elements: $(f\ a\ b\ c)$. Similarly, a nested invocation like $LCM(GCD(72, 15), 4)$ is represented by a nested list structure (S-expression): $(LCM\ (GCD\ 72\ 15)\ 4)$.

Function calls are evaluated by evaluating first their arguments. Symbols and S-expressions prefixed with a single quote (`'`) evaluate to themselves, i.e. are supplied literally to the invoked function. Numeric arguments also evaluate to themselves, while unquoted symbols evaluate to the value they are *bound* to, a concept explained later.

However, if any argument itself is a non-quoted S-expression, it is considered as an invocation of another function (or special-form), and is evaluated before its value can be used in its place, as an argument to an original function. Thus, if an S-expression is viewed as a rooted general plane tree, its evaluation triggers a depth-first traversal down to its quoted branches and the symbolic and numeric leaves.

Note that the special forms differ from real functions in that not all of their “arguments” are evaluated at all the times.

The following list gives the built-in functions and special forms that are sufficient to implement most of the Catalan automorphisms mentioned in this paper.

`(cons Sexpr1 Sexpr2)`

Returns a new S-expression $(Sexpr1 . Sexpr2)$

Remark. In the context of binary trees (c/d) , `cons` creates a new binary tree from the left hand side tree $Sexpr1$ and the right hand side tree $Sexpr2$. In the context of general trees (e) , `cons` creates a general tree, by grafting tree $Sexpr1$ as a scion to the left of all the branches of tree $Sexpr2$. In the context of Dyck paths (i) , `cons` creates a new Dyck path by inserting to ...

Examples:

<code>(cons 'a 'b)</code>	\rightarrow	<code>(a . b)</code>	
<code>(cons '() '(() . ()))</code>	\rightarrow	<code>((() . ((() . ())))</code>	[Result in internal, dotted-pair format.]
	\rightarrow	<code>((() ()))</code>	[Result in external format.]

`(car Pair)`

Pair should be of the form $(Sexpr1 . Sexpr2)$. This function returns *Pair*'s left-hand side, i.e. *Sexpr1*.

²Most of what is said here applies also to LISP, with just minor changes in the names of some primitives and functions. We might gloss over some details, as our purpose is just to describe a subset of the language sufficient for implementing Catalan bijections.

Examples:

(car '(a b)) → a
(car '((a b) c d)) → (a b)

(cdr *Pair*)

Pair should be of the form (*Sexpr1* . *Sexpr2*). This function returns *Pair*'s right-hand side, i.e. *Sexpr2*.

Examples:

(cdr '(a b)) → (b)
(cdr '((a b) c d)) → (c d)

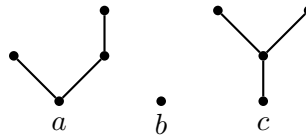
(list *Sexpr1* .. *Sexpr_n*)

Returns a list constructed from its arguments.

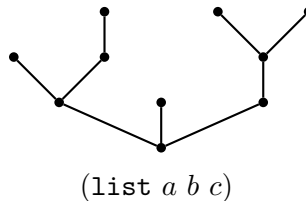
Examples:

(list) → () [With no arguments gives an empty list.]
(list '(a b)) → ((a b)) [Unary form surrounds with extra parentheses.]
(list 'a 'b) → (a b)

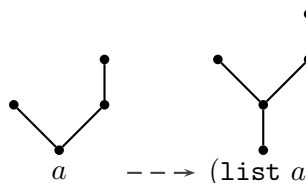
Remark. In the context of general trees (*e*), `list` constructs a new tree, by making all the argument trees children of a new root. For example, if we have trees *a*, *b* and *c* (note that tree *b* is an empty tree, corresponding to S-expression ()):



then `(list a b c)` constructs a new tree whose root vertex has degree three (i.e. has three child-trees):



In the context of general trees the unary form constructs a new general tree, by “planting” the argument tree above the trunk of one edge:



In the context of Dyck paths (i), unary `list` rises its argument onto a "pedestal": $\nearrow \cdots \searrow$, so we have $(\text{list } \nearrow \searrow) = \nearrow \searrow$ for example.

In the context of parenthesizations (P), an unary `list` surrounds its argument with extra parentheses.

In the context of binary trees (c/d), $(\text{list } d)$ creates a new binary tree, whose left hand side tree will be d and the right hand side tree an empty leaf, so we have $(\text{list } \swarrow \searrow) = \swarrow \searrow$ for example.

`(null? Sexpr)`

Returns `#t` (true) if its argument is an empty list `()`, otherwise `#f` (false).

Examples:

```
(null? '(a))    → #f
(null? (list))  → #t
```

`(pair? Sexpr)`

Returns `#t` (true) if its argument is a cons cell, otherwise `#f` (false).

Examples:

```
(pair? '(a))    → #t
(pair? (list))  → #f
```

`(not Sexpr)`

Returns `#t` (true) if its argument is `#f` (false), and for all other values returns `#f` (false).

Examples:

```
(not (pair? '(a))) → #f
(not (pair? (list))) → #t
```

Note: As long we restrict our attention to *Nihilistic S-expressions* (as in the implementations of various Catalan automorphisms in this paper), then

`(null? Sexpr)` is equivalent to `(not (pair? Sexpr))`

and

`(pair? Sexpr)` is equivalent to `(not (null? Sexpr))`

`(if Sexpr1 Sexpr2 Sexpr3)`

Returns the value of `Sexpr2` if `Sexpr1` evaluates to a non-false value, otherwise evaluates `Sexpr3` and returns its value. Note that `if` always evaluates either `Sexpr2` or `Sexpr3`, but never both.

Examples:

```
(if (null? (list)) (list 'it 'is 'empty) (list 'not 'empty)) → (it is empty)
(if (pair? (list)) (list 'it 'is 'a 'pair) (list 'not 'pair)) → (not pair)
```

```
(cond (TestExprA ExprA1 ... ExprAn)
      (TestExprB ExprB1 ... ExprBn)
      ...
      (else ExprElse1 ... ExprElsen)
)
```

If *TestExprA* evaluates to a non-false value, executes expressions *ExprA₁ ... ExprA_n* listed after it, returning the value of the last one (*ExprA_n*) as the value of whole **cond**-expression, without evaluating any other expressions. Otherwise, checks whether *TestExprB* evaluates to a non-false value, and if so, executes expressions *ExprB₁ ... ExprB_n* listed after it, again returning the value of the last one as the value of the whole **cond**-expression. If none of the test-expressions evaluate to a non-false value before **else**-branch, then the expressions *ExprElse₁ ... ExprElse_n* are evaluated, with the value of the last one returned as the value of the whole **cond**-expression.

Examples:

```
(cond ((null? (list)) (list 'it 'is 'empty))
      (else (list 'it 'is 'not 'empty)))
)
→ (it is empty)
```

Note:

(**if** *Expr₁ Expr₂ Expr₃*) is equivalent to (**cond** (*Expr₁ Expr₂*) (**else** *Expr₃*))

```
(define (funcname arg1 ... argn)
  expr1
  ...
  exprn
)
```

Defines a new function named *funcname* with formal arguments *arg₁ ... arg_n*. When the function *funcname* is invoked, the values of the actual arguments will be bound to the formal arguments, and each of the expressions *expr₁ ... expr_n* will be executed in turn, with the value of *expr_n* being returned as the result of the function invocation.

Examples:

```
(define (ourlcm a b) (* a (/ b (gcd a b))))
[We define our version of
the LCM-function, using the
built-in functions "*" (multi-
plication), "/" (integer divi-
sion) and "gcd".]
(ourlcm 12 15) → 60 [Works as expected.]
```

```
(let ((Sym1 InitExpr1)
      ...
      (Symn InitExprn))
  Expr1
  ...
  Exprn
)
```

Evaluates the expressions *InitExpr₁ ... InitExpr_n* in parallel, and binds the symbols *Sym₁ ... Sym_n* temporarily to the resulting values, then executes each of the expressions *Expr₁ ... Expr_n* in turn, returning the value of the last one (*Expr_n*) as the result of the whole **let**-expression. Note that in SCHEME **let** offers the standard way to define temporary variables inside the functions definitions.

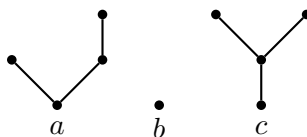
(append $Sexpr_1 \dots Sexpr_n$)

Returns a list concatenated from its arguments.

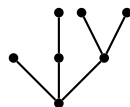
Examples:

(append)	→	()	[With no arguments gives an empty list.]
(append '(a b) '(c))	→	(a b c)	
(append '() '(a b) '() '(c d) '())	→	(a b c d)	[Empty lists are silently ignored.]





Remark. In the context of general trees (e), **append** creates a new general tree, by connecting them from their root vertices in the same left to right order. For example, if we have trees a , b and c (note that tree b is an empty tree, corresponding to S-expression ()):



then (append $a b c$) will just silently discard the argument b as it is empty, and identify (XXX – Terminology?) the roots of trees a and c with each other. We see that the degree of the new tree is the sum of the degrees of the argument trees, in this case $2 + 0 + 1 = 3$.



(append $a b c$)

In the context of Dyck paths (i), **append** just concatenates the structures given as its arguments, so we have (append   ) =  for example.

Note: We can implement our own, a strictly 2-ary version of **append** with the following recursive definition:

```
(define (append2 a b)
  (cond ((null? a) b)
        (else (cons (car a) (append2 (cdr a) b))))
)
```

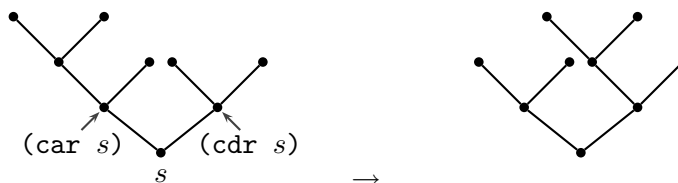
4. Programming reflections & rotations of common interpretations with Scheme

With information I have given so far, it is now easy to define a couple of simple automorphisms and other bijections. I will later give more formal definition for each, but for now, this gives the reader some taste how the system works.

For example, **A069770* is a bijection which simply swaps the left and right-hand sides of a binary tree with each other:

```
(define (*A069770 s)
  (if (null? s) s [If s is ( ), just return it back.]
      (cons (cdr s) (car s)) [Otherwise, return a new S-
                             expr cell constructed from car
                             and cdr of the old cons cell,
                             but in exchanged order.]
    )
)
```

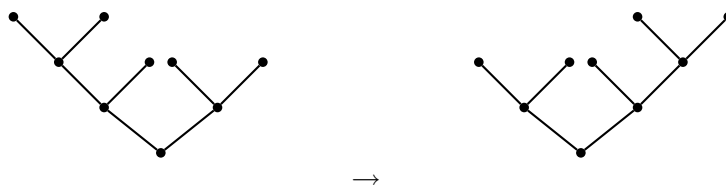
Thus, for a binary tree shown earlier, **A069770* transforms it in the following manner:



Now, what if we, instead of just swapping the left and right hand side of a binary tree, wanted to get its mirror-image? This can be done by recursively swapping the children of *all the nodes* of the tree, and bijection **A057163* does just that:

```
(define (*A057163 s)
  (if (null? s) s [If s is ( ), just return it back.]
      (cons (*A057163 (cdr s)) (*A057163 (car s))) [Otherwise, return a new S-
                                                      expr cell constructed from the
                                                      recursively reflected cdr and
                                                      car elements of the original
                                                      cell.]
    )
)
```

Thus, **A057163* transforms our example binary tree in this way:



As there is a very direct isomorphism between binary trees and polygon triangulations (a), we see that **A057163* also reflects the latter interpretation, thus we have $(*A057163 \text{ } \heartsuit) = \heartsuit$ for example, corresponding exactly to the binary trees shown above.

Now consider the following function definition:

```
(define (*A057509 s)
  (if (null? s) s [If s is ( ), just return it back.]
      (append (cdr s) (list (car s))) [Otherwise, return the tail of
                                       the original list on whose end
                                       the original head is concate-
                                       nated to.]
    )
)
```

So what this

does? In the context of lists, it rotates the top-level structure of the list one element leftward, moving the original first element ("head of the list") to be the last element of the new list. In the context of general trees (e), we have $(*A057509 \ \downarrow \downarrow \downarrow) = \downarrow \downarrow \downarrow$ for example, and with Dyck paths (i), Note that if the tree is "planted", i.e. of degree one, then it is not changed by $*A057509$, thus $(*A057509 \ \downarrow) = \downarrow$ for example.

Now, let's add a little recursive twist to above definition, to get bijection $*A057508$:

```
(define (*A057508 s)
  (if (null? s) s [If s is (), just return it back.]
      (append (*A057508 (cdr s)) (list (car s))))
)
```

What this function might do? It moves the head of the original list to the end position as before. The difference is that now the tail of the original list is treated in the same way, recursively. Now, the funny thing with recursive functions is, that they are much easier to grasp, if we *already* know what they do, or at least if we can guess it (i.e. make a sensible induction hypothesis), So, after telling that this reverses the top-level of the list, it is now easy to see that it indeed does that: a list on the end of whose reversed tail portion the original first element is appended to, is clearly the same thing as the whole list reversed!

So, we might as well have defined this as

```
(define (*A057508 s) (reverse s))
```

as that is precisely what the built-in function `reverse` does in Lisp and Scheme.

In the context of general trees (e), we have $(*A057508 \ \downarrow \downarrow \downarrow) = \downarrow \downarrow \downarrow$ for example, and with Dyck paths (i), As with $*A057509$, if the tree is "planted", i.e. of degree one, then it is neither changed by $*A057508$, thus $(*A057508 \ \downarrow) = \downarrow$ and equally, when the Dyck path is on a "pedestal", as it is analogous case: $(*A057508 \ \nearrow \nearrow \nearrow \searrow) = \nearrow \nearrow \nearrow \searrow$ for example.

What if we want to *deepreverse* a general tree, that is, to reverse it on *every* level, not just the immediate children of the root vertex? In that case we just add a recursion call also to car-branch, and we get bijection $*A057164$:

```
(define (*A057164 s)
  (if (null? s) s
      (append (*A057164 (cdr s)) (list (*A057164 (car s)))))
)
```

Now we have $(*A057164 \ \downarrow) = \downarrow$ and equally, in the context of Dyck paths:
 $(*A057164 \ \nearrow \nearrow \nearrow \searrow) = \searrow \searrow \searrow \nearrow$ for example. So, what $*A057164$ does, is that it reflects the general trees, Dyck paths and parenthesizations. Because of this, it is actually *easier* to define in terms of balanced binary sequences, than in terms of S-expressions.

It also reflects the interpretation (n) (noncrossing handshakes) and we have $(*A057164 \ \circlearrowleft) = \circlearrowright$ and vice versa.

What if we wanted to *rotate* noncrossing handshakes, say one step counterclockwise? What kind of function could do that? The answer is surprisingly simple:

```

(define (*A057501 s)
  (if (null? s) s
      (append (car s) (list (cdr s)))
  )
)

```

XXX - Needs a graphical explanation, at least! We see that the definition is exactly like that of **A057509*, except that call to `append` has now calls to `car` and `cdr` in another order, and indeed, we can as well define bijection **A057501* as a composition of already defined bijections **A057509* and **A069770*:

$$*A057501 = *A057509 \circ *A069770 \tag{1}$$

We keep to the convention that in compositions the rightmost element acts first, i.e. that the above definition is equivalent to

```

(define (*A057501 s) (*A057509 (*A069770 s)))

```

And we see that $(*A057501 \textcircled{\circ}) = \textcircled{\ominus}$ for example, and furthermore $(*A057501 \textcircled{\ominus}) = \textcircled{\circ}$, $(*A057501 \textcircled{\omin�}) = \textcircled{\omin�}$, $(*A057501 \textcircled{\omin�}) = \textcircled{\omin�}$, $(*A057501 \textcircled{\omin�}) = \textcircled{\omin�}$, $(*A057501 \textcircled{\omin�}) = \textcircled{\omin�}$ and $(*A057501 \textcircled{\omin�}) = \textcircled{\omin�}$, after which we have come a full circle.

As the interpretations(*n*) and (*e*) are closely related, it is interesting to see what happens with the corresponding general trees: Then $(*A057501 \Uparrow) = \Downarrow$, $(*A057501 \Downarrow) = \Uparrow$, $(*A057501 \Uparrow) = \Downarrow$, $(*A057501 \Downarrow) = \Uparrow$, $(*A057501 \Uparrow) = \Downarrow$ and $(*A057501 \Downarrow) = \Uparrow$, after which we have again come a full circle. We see that in the context of general trees (*e*), the automorphism **A057501* keeps the tree intact (*as a planar tree*), but only rotates the the position of the root, to the first vertex left of the current root.

How to rotate interpretation (*a*) (polygon triangulations) then? We add just one recursion call (*to the car-branch*) into the definition of **A057501*, and we get:

```

(define (*A057161 s)
  (if (null? s) s
      (append (*A057161 (car s)) (list (cdr s)))
  )
)

```

And we see that $(*A057161 \textcircled{\omin�}) = \textcircled{\omin�}$ for example, i.e. **A057161* rotates polygon triangulations counterclockwise. XXX - Needs a graphical explanation, at least!

5. Using destructive Scheme-primitives to implement Catalan bijections

Rationale. In the previous section we used a built-in Scheme-function `append` in most of the examples. However, analyzing such definitions is often unwieldy, and also, defining inverse bijections for such non-involutive automorphisms as rotations **A057509*, **A057501* and **A057161* is not easy/expedient with by using `append`. Many bijections are actually easier to define using so called *destructive* (or physical) list-operations. The fundamental primitives for doing this kind of operations in SCHEME are called `set-car!` and `set-cdr!`³ Their definitions are given below.

```

(set-car! Pair SexprNew)

```

³Known in LISP as `rplaca` and `rplacd`.

Modifies physically the left-hand side of *Pair* which should be of the form $(Sexpr1 . Sexpr2)$, and which after the modification is transformed to $(SexprNew . Sexpr2)$. An error results if the first argument is not a cons cell.

Examples:

```
(let ((ourpair (list 'a 'b)))      [ourpair is bound to (a b)]
      (set-car! ourpair 'c)
      ourpair
  )
  →                               (c a)
```

`(set-cdr! Pair SexprNew)`

Modifies physically the right-hand side of *Pair* which should be of the form $(Sexpr1 . Sexpr2)$, and which after the modification is transformed to $(Sexpr1 . SexprNew)$. An error results if the first argument is not a cons cell.

Examples:

```
(let ((ourpair (list 'a 'b)))      [ourpair is bound to (a b)]
      (set-cdr! ourpair '(c))
      ourpair
  )
  →                               (a c)
```

Definition. We say that a Scheme implementation of Catalan automorphism is *destructive* if it invokes either of the primitives `set-car!` or `set-cdr!` defined above, or any function which invokes them explicitly or implicitly. If a Scheme implementation of a Catalan automorphism is not destructive, then we say that it is *constructive*.

If we exclude `set-car!` and `set-cdr!` from the set of primitives listed above, we can only define automorphisms and other functions that leave intact the physical structure of the S-expression(s) given as their argument(s). Apart from the trivial identity automorphism, on some values of its argument every Catalan automorphism must return an S-expression with different structure than the argument has. Because by its very definition a Catalan automorphism is *a bijection on the set of structures of the same size* it is impossible to implement this with a function containing only invocations of the accessor-functions `car` and/or `cdr`, as then the result's size (*in cons cell nodes*) would be less than that of the argument. So to effect changes which return an S-expression of the same size but of different structure, it must invoke the function `cons`, or some other functions like `append` or `list` which calls `cons` implicitly. For this reason these are called *constructive* implementations of Catalan automorphisms.

On the other hand, if an implementation of an automorphism invokes `set-car!` and/or `set-cdr!` or any other function invoking them, it is called *destructive*. This is because such functions make direct modifications to their argument's physical structure, and thus destroy information about their original structure. Traditionally, use of such destructive functions has been considered somewhat dangerous, and thus SCHEME has adopted the convention of suffixing the names of such functions with an exclamation mark (!). For example, there is a function called `append!` which is a destructive version of the function `append` described above. However, it turns out that in our application the use of strictly destructive implementations of Catalan automorphisms (*or functions of suspected of being isomorphic*) have certain merits, when they are subjected to mathematical analysis. Namely, the bijectivity of such a function is much easier to prove, if we know that it neither does "lose" any existing cons cells, nor allocate any new ones with constructive operations. More about this in the last section of this paper. For most of the automorphisms represented

below we thus give both constructive and destructive definition. We follow the Scheme practice of suffixing the names of the latter ones with '!'.

For example, here is a destructive version of automorphism **A069770* whose constructive version was defined above. The comments given in brackets clarify the steps.

```
(define (*A069770! s)
  (if (null? s) s [If s is (), then return it
                  back.]
      (let ((org-car (car s))) [Otherwise, store its car-side
                                to org-car]
          (set-car! s (cdr s)) [before it is overwritten with
                                cdr-side of s.]
          (set-cdr! s org-car) [Then overwrite the cdr-side
                                with the original car-side of
                                s,]
          s [and return the modified s as
            the value of the whole func-
            tion.]
      )
  )
)
```

Now the destructive implementation of automorphism **A057163* can be written simply as:

```
(define (*A057163! s)
  (cond ((pair? s)
         (*A069770! s)
         (*A057163! (car s))
         (*A057163! (cdr s))
        )
        s
  )
)
```

(2)

that is, we apply automorphism **A069770* to every cons cell we encounter (i.e. swap their sides), and recurse down to both car- and cdr-side of each node, continuing down each branch as long as we encounter ().

6. Integer-Sequences Associated with Catalan bijections

We are interested about several sequences of integers that each Catalan automorphism naturally produces. Some of these provide merely fingerprint information about a particular automorphism, suitable for recording into such online databases as Neil J.A. Sloane's OEIS (*Online Encyclopedia of Integer Sequences*) [Sloane 1995–], while others may have intrinsic interest of their own.

First we define a few auxiliary functions that turn out useful in later definitions. As their names we use the A-numbers with which they have been submitted to OEIS.

Definition 6 (A014137 and A014138) A014137 gives the partial sums of Catalan numbers (A000108), starting the summing from A000108(0), yielding

1, 2, 4, 9, 23, 65, 197, 626, 2056, 6918, 23714, ...

while A014138 starts the summing from A000108(1), yielding

$$1, 3, 8, 22, 64, 196, 625, 2055, 6917, 23713, \dots$$

Note that if we organize all Catalan structures (of some interpretation) in a sequence ordered by their size, so that we first have a single empty structure of size 0 at index 0, followed by a single structure of size 1 at index 1, followed by two structures of size 2 at indices 2 and 3, followed by five structures of size $n=3$, etc. with each subsequence containing $C(n)$ structures of size n , then $[A014137(n-1)..A014138(n-1)]$ gives the inclusive limits for the structures of size n in such a sequence. (For the convenience we can assume that $A014137(-1) = A014138(-1) = 0$.)

Definition 7 (A014486) This is a sequence which contains 0 at index 0 (corresponding to an empty structure), and thereafter in each subrange $[A014137(n-1)..A014138(n-1)]$ binary-coded representations of all A000108(n) Dyck-paths/parenthesizations of size n sorted into lexicographic order. We get a binary-coded representation from a Dyck-path/parenthesization by mapping each upward-slope/left parenthesis to 1, and each downward-slope/right parenthesis to 0, and then reading the resulting number as it were a binary number. This is illustrated in the table 3.

The sequence begins as

$$0, 2, 10, 12, 42, 44, 50, 52, 56, 170, 172, 178, 180, 184, 202, 204, 210, 212, 216, 226, 228, 232, 240, \dots$$






n	(i)	A063171(n)	A014486(n)
1.		10 in binary	2 in decimal
2.		1010 in binary	10 in decimal
3.		1100 in binary	12 in decimal
4.		101010 in binary	42 in decimal
5.		101100 in binary	44 in decimal

Table 3: Example showing how the first few terms of A014486 are formed.

Definition 8 (A080300) This sequence is the inverse function of A014486. For those n which do not occur as values of A014486 it gives zero, otherwise n 's position in A014486. That is, we have $n = A080300(A014486(n))$ for all n .

The sequence begins as:

$$0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 3, 0, 0, 0, 0, \dots$$

Definition 9 (binexp→Sexp and Sexp→binexp) These functions convert between the terms of A014486 (i.e. totally balanced binary sequences) and S-expressions. They are each other's inverses, i.e. we have $s = \mathbf{binexp} \rightarrow \mathbf{Sexp}(\mathbf{Sexp} \rightarrow \mathbf{binexp}(s))$ for all symbolless S-expressions s .

The SCHEME-code for their implementation is given in the Appendix P.

Definition 10 (Signature permutation) The Signature Permutation $sp_g(0), sp_g(1), sp_g(2), sp_g(3), \dots$ of automorphism g is a sequence $sp_g(n), n = 0..∞$ where

$$sp_g = \mathbf{A080300} \circ \mathbf{Sexp} \rightarrow \mathbf{binexp} \circ g \circ \mathbf{binexp} \rightarrow \mathbf{Sexp} \circ \mathbf{A014486}$$

Remark. Here it is assumed that automorphism g has been specified as a function whose domain and range are S-expressions, so we need the functions $\mathbf{binexp} \rightarrow \mathbf{Sexp}$ and $\mathbf{Sexp} \rightarrow \mathbf{binexp}$ to do a round-trip from A014486-codes and back.

Definition 11 (Set S of signature permutations) In this paper the set of signature-permutations of all possible Catalan automorphisms is called the set S. This set consists of all such permutations of natural numbers where no cycle resides in more than one subrange [A014137(n)..A014138(n)].

Clearly this set of permutations is closed with respect to composition and taking inverses, the former operation corresponding to the composition of two automorphisms and the latter with finding the inverse of an automorphism. Furthermore, the OEIS sequence A001477 (The nonnegative integers) works as the identity element of this permutation group. Thus everything regarding automorphisms could be in principle expressed as group operations of this group S, which itself is a subgroup of S_∞ .

Remark. The set S is not enumerable, which can be seen with the help of standard diagonal argument. However, it turns out that there are important subgroups of the set S that are enumerable.

Definition 12 (Cycle-Count Sequence) The Cycle-Count Sequence $cc_g(0), cc_g(1), cc_g(2), cc_g(3), \dots$ of automorphism g is a sequence $cc_g(n), n = 0..∞$ where $cc_g(n)$ gives the number of orbits into which the set $\{C_n$ structures of size $n\}$ is partitioned under the action of g .

That is, $cc_g(n)$ is equal to number of cycles in the subrange [A014137($n-1$)..A014138($n-1$)] of the signature permutation of g .

Conceptually, Cycle-Count sequence gives the number of Catalan structures of size n "up to automorphism" g .

Definition 13 (counts of fixed points sequence) The counts of fixed points sequence $fcs_g(0), fcs_g(1), fcs_g(2), \dots$ of automorphism g is a sequence $fcs_g(n), n = 0..∞$ where

$$fcs_g(n) = \#\{i \in [A014137(n-1)..A014138(n-1)] \mid sp_g(i) = i\}$$

That is, $fcs_g(n)$ gives the number of structures in the set $\{C_n$ structures of size $n\}$ that are fixed (stay same) under the action of g .

Conceptually, counts of fixed points sequence gives the number of Catalan structures of size n that are "symmetric" with respect to automorphism g .

Definition 14 (Max-Cycle Sequence) The Max-Cycle Sequence $max_g(0), max_g(1), max_g(2), max_g(3), \dots$ of automorphism g is a sequence $max_g(n), n = 0..∞$ where $max_g(n)$ gives the size of the a largest orbit into which the set $\{C_n$ structures of size $n\}$ is partitioned under the action of g , that is, the size of a largest cycle in the in the subrange $[A014137(n-1)..A014138(n-1)]$ of the signature permutation of g .

Definition 15 (LCM Sequence) The LCM Sequence $lcm_g(0), lcm_g(1), lcm_g(2), lcm_g(3), \dots$ of automorphism g is a sequence $lcm_g(n), n = 0..∞$ where $lcm_g(n)$ gives the *least common multiple* of all orbit sizes into which the set $\{C_n$ structures of size $n\}$ is partitioned under the action of g , that is, it is equal to the least common multiple of all cycle sizes present in the subrange $[A014137(n-1)..A014138(n-1)]$ of the signature permutation of g .

Remark. Of the above five sequence-types, only the signature-permutation offers a unique fingerprint for each Catalan automorphism, while the other four, although they might not distinguish two automorphisms from each other, at least give some hints about their possible relationships. Especially with automorphisms that are *involutions* (that is, self-inverse), the counts of fixed points sequence alone specifies each (such automorphism) up to all of its conjugations. Even if we have two distinct, non-involutive automorphisms, we should have a strong suspicion that they are conjugates of each other if their cycle-count, fix-point, max-cycle and lcm sequences appear to match term by term.

6.1. Using "Burnside's Lemma" to compute Cycle-Count sequence.

Burnside's lemma, sometimes also called Cauchy-Frobenius or just Orbit Counting lemma, states that the number of orbits into which the set X is partitioned by group G (whose members act on X 's elements) is equal to the average number of points fixed by the elements of G :

$$\frac{1}{|G|} \sum_{g \in G} \#\{x \in X \mid g(x) = x\}$$

In our case, the set X is the set of $\{C_n$ structures of size $n\}$ and the group G is a cyclic group consisting of automorphism g and its successive powers g^2, g^3, g^4, \dots . When acting on any finite sized set X , the group is in practice finite, as there exists an integer u such that $g^u(x) = x$ for all x in the set of $\{C_n$ structures of size $n\}$ thus g^u is equal to an identity element of group G , and all higher powers of g can be taken as modulo u .

The integer sequence **A078491** is defined as the least common multiple of all natural numbers from 1 to n th Catalan number. It grows with a formidable rate:

1, 1, 2, 60, 360360, 219060189739591200, 1749342047920660916901891145781670987072592322134428432000, ...

Setting $u = A078491(n)$ we can be sure that g^u is an identity element (neutral permutation) for *any* automorphism g acting on the set of size n . However, to use Burnside's lemma in practice, we need more down-to-earth bounds for u . Indeed, for each particular automorphism g , it is just the sequence $lcm_g(n)$ which offers us the actual lower bound (???) of u when g acts on the set of size n .

Applying Burnside's lemma, the Cycle-Count sequence of automorphism g can be calculated with the formula:

$$cc_g(n) = \frac{1}{lcm_g(n)} \sum_{i=1}^{lcm_g(n)} fcs_{g^i}(n) \quad (3)$$

which requires that we also know how to calculate $fcs_{g^i}(n)$, i.e. the counts of fixed points sequence for each power g^i of automorphism g up to $i = lcm_g(n)$, and also have a method for calculating the lcm-sequence $lcm_g(n)$, or some sequence $b(n)$, for which it holds that

$$\forall n, lcm_g(n) | b(n)$$

(EDELLINEN KOHTA PAREMMIN!)

However, in many cases these sequences grow too fast that the calculation with Burnside's lemma would be any faster than the explicit counting of the orbits with a computer program, unless the formula is amenable to further reduction.

7. Embeddability

Definition 16. We say that automorphism f *embeds into* automorphism g *in scale* $n : m$ if there is such an injection:

$$e : \{C_n \text{ structures of size } n\} \rightarrow \{C_m \text{ structures of size } m\}$$

that for all elements s of the domain set, it holds that

$$e(f(s)) = g(e(s)). \quad (4)$$

As e is an injection, it has a well-defined inverse, and the above identity can be also written as:

$$f = e^{-1} \circ g \circ e \quad (5)$$

7.1. Examples of embeddability.

Definition 17. We say that automorphism g is *Lukasiewicz-word permuting*, if for all parenthesizations s , the Lukasiewicz-word of $g(s)$ is always a permutation of the Lukasiewicz-word of s itself. More formally, automorphism g is Lukasiewicz-word permuting if its signature permutation satisfies

$$A129593(sp_g(n)) = A129593(n) \quad (6)$$

for all $n \geq 0$.

Consider that subset of general trees whose Lukasiewicz-words contain only digits '0' and '2' in some order. These trees are isomorphic with plane binary trees, although now occurring amongst plane general trees whose internal binary tree representation contains twice as many vertices. Thus, if g is Lukasiewicz-word permuting, then that subset is closed under the action of g , and g 's restriction to that subset induces another automorphism f . Specifically, in that case, the following holds

$$f = *A083927 \circ g \circ *A057123 \quad (7)$$

Here $*A057123$ is used as an injection e of definition (5), and it specifically maps each binary tree to a corresponding general tree of the same shape. $*A083927$ is its inverse.

Rephrasing this in more general terms, we obtain:

Lemma 1. *For any Lukasiewicz-word permuting automorphism g there is an automorphism f which embeds into automorphism g in scale $n : 2n$.*

Definition 18. We say that automorphism g preserves the *initial nil* if for all S-expressions s it holds that

$$(g (\text{cons } ' () s)) = (\text{cons } ' () (f s)) \quad (8)$$

where the function f is either the same or different automorphism than g . In other words, such an automorphism preserves the second to left zero in the Lukasiewicz-word of an S-expression, if present.

Definition 19. We say that automorphism g keeps *planted trees as planted* if for all parenthesizations s it holds that

$$(g (\text{cons } s ' ())) = (\text{cons } (f s) ' ()) \quad (9)$$

where the function f is either the same or different automorphism than g . In other words, such an automorphism preserves the initial 1 in the Lukasiewicz-word of an S-expression, if present.

Definition 20. We say that automorphism g is *horizontally telescoping* if for all parenthesizations s it holds that

$$(g (\text{cons } ' () s)) = (\text{cons } ' () (g s)) \quad (10)$$

that is, the injection e required by the general definition of embeddability defined in (4) is now defined as:

$$(\text{define } (e s) (\text{cons } ' () s)) \quad (11)$$

We note that this is a special case of a weaker condition given in (8), realized when $f = g$,

In this case the injection e maps the parenthesizations of size n to the parenthesizations of size $n + 1$ by simply concatenating empty parentheses *to the front* of the corresponding parenthesization.

This is equivalent to saying that each sub-permutation of the length A000108(n) induced by automorphism g 's action on the standard sequence of lexicographically ordered parenthesizations (A014486) starts with the same cycle-structure as the previous sub-permutation. In this case we can form yet another permutation of the natural numbers by conceptually taking the "infinith" of such sub-permutations and by "normalizing" it to begin from 0 or 1.

Definition 21. We say that automorphism g is *vertically telescoping* if for all parenthesizations s it holds that

$$(g (\text{cons } s ' ())) = (\text{cons } (g s) ' ()) \quad (12)$$

where the injection e required by the general definition of embeddability is now defined as:

$$(\text{define } (e s) (\text{cons } s ' ())) \quad (13)$$

That is, the injection e is equal to the unary variant of SCHEME-function `list`, which maps the parenthesizations of size n to the parenthesizations of size $n + 1$ by surrounding them with one extra pair of parentheses.

We note that this is a special case of a weaker condition given in (9), realized when $f = g$,

Definition 22. The above two definitions are examples of *self-embeddable* automorphisms.

We say that automorphism g is *self-embeddable* if g embeds into itself in scale $n : m$, where n ranges through all values in \mathbb{N} , and m is a function of n , with $m > n$.

Notes. The allusions "*horizontal*" and "*vertical*" should make sense when one thinks in the terms of Dyck paths (Catalan Mountain Ranges), and also the plane general trees in the latter case.

If a horizontally telescoping automorphism is conjugated with automorphism **A057163* we obtain a vertically telescoping automorphism, and *vice versa*. A Lukasiewicz-word permuting automorphism can also be a vertically telescoping automorphism, which implies that it must preserve the initial 1 of the Lukasiewicz-word when present. (In other words, keeps planted trees as planted.)

It should be obvious that all self-embeddable automorphisms of the scale $n : n+1$ have genuinely monotone (growing) cycle count sequences and monotone (but not necessarily genuinely) fix point sequences from $n \geq 1$ onward.

8. Recursion Schemata

We list several recursion schemes by which to construct new automorphisms from old ones.

Definition 23 (FORK: Apply at root, recurse into both branches.) In this recursion scheme, the given automorphism is applied at the root node, and then the same process is repeated recursively for both car- and cdr-branch of the S-expression.

In SCHEME we can define the following *higher order function* that takes as its argument a destructively defined function `foo!` and returns back a new function `bar!`, that also acts on S-expressions:⁴

```
(define (!FORK foo!)
  (letrec ((bar! (lambda (s)
                  (cond ((pair? s)
                        (foo! s)
                        (bar! (car s))
                        (bar! (cdr s)))
                        )
                  )
          s
        ))
    bar!
  )
)
```

(14)

Note that this function uses the variant `letrec` of `let`, which allows the definition of recursive lambda-forms (*anonymous functions*). Calling this functional as `(!FORK *A069770!)` produces the same destructive definition of automorphism **A057163!* that was given at the end of section 4.

Definition 24 (KROF: Recurse into both branches, then apply at root.) In this recursion scheme, the given automorphism is applied at the root node, but only *after* the same process has first been repeated recursively for both car- and cdr-branch of the S-expression.

⁴In this paper the names of all higher order functions expecting a destructively implemented automorphism as their function argument are *prefixed* with an exclamation mark (!).

This is defined in SCHEME as:

```
(define (!KROF foo!)
  (letrec ((bar! (lambda (s)
                 (cond ((pair? s)
                        (bar! (car s))
                        (bar! (cdr s))
                        (foo! s)
                        )
                 )
          s
          )
          ))
  bar!
)
```

 (15)

Definition 25 (SPINE: Apply at root, then recurse into cdr branch.) In this recursion scheme, the given automorphism is applied at the root node, and then the same process is repeated recursively for the cdr-branch of the S-expression.

This is defined in SCHEME as:

```
(define (!SPINE foo!)
  (letrec ((bar! (lambda (s)
                 (cond ((pair? s)
                        (foo! s)
                        (bar! (cdr s))
                        )
                 )
          s
          )
          ))
  bar!
)
```

 (16)

Definition 26 (ENIPS: Recurse into cdr branch, then apply at root.) In this recursion scheme, the given automorphism is applied at the root node, but only *after* the same process has first been repeated recursively for the cdr-branch of the S-expression.

This is defined in SCHEME as:

```
(define (!ENIPS foo!)
  (letrec ((bar! (lambda (s)
                 (cond ((pair? s)
                        (bar! (cdr s))
                        (foo! s)
                        )
                 )
          )
          s
          ))
    bar!
  )
)
```

(17)

Definition 27 (RIBS: Apply at car branch, then recurse into cdr branch.) In this recursion scheme, the given automorphism is applied at the car branch, and then the same process is repeated recursively for the cdr-branch of the S-expression.

This is defined in SCHEME as:

```
(define (!RIBS foo!)
  (letrec ((bar! (lambda (s)
                 (cond ((pair? s)
                        (foo! (car s))
                        (bar! (cdr s))
                        )
                 )
          )
          s
          ))
    bar!
  )
)
```

(18)

Alternatively, RIBS could be defined using the built-in SCHEME primitive (`for-each f! l`), which applies a destructive function $f!$ to each top-level element of the list l .

```
(define (!RIBS foo!)
  (letrec ((bar! (lambda (s)
                 (for-each foo! s)
                 )
          )
          s
          ))
    bar!
  )
)
```

(19)

Definition 28 (DEEPEN: Apply at root, then recurse into each car branch.) In this recursion scheme, the given automorphism is applied at the root node, and then the same process is repeated recursively for each car-branch of the S-expression.

This is defined in SCHEME as:

```
(define (!DEEPEN foo!)
  (letrec ((bar! (lambda (s)
                 (cond ((pair? s)
                        (foo! s)
                        (for-each bar! s)
                        )
                        )
                 )
          ))
    bar!
  )
)
```

(20)

Definition 29 (NEPEED: Recurse into each car branch, then apply at root.) In this recursion scheme, the given automorphism is applied at the root node, but only *after* the same process has first been repeated recursively for each car-branch of the S-expression.

This is defined in SCHEME as:

```
(define (!NEPEED foo!)
  (letrec ((bar! (lambda (s)
                 (cond ((pair? s)
                        (for-each bar! s)
                        (foo! s)
                        )
                        )
                 )
          ))
    bar!
  )
)
```

(21)

Note that DEEPEN and NEPEED convert certain *shallow* automorphisms to the corresponding *deep* variants.

Proposition 1. *Whenever any of the above recursion schemes is applied to a destructively defined automorphism, the resulting function is also a destructively defined automorphism.*

Proof. Because no constructive functions are involved, we can be sure that the resulting function does not map any S-expression to a larger structure. Thus, to show the bijectivity we just need to demonstrate that any derivative function defined with these recursion schemas has an inverse.

Indeed, the following four identities hold:

$$(!FORK f)^{-1} = (!KROF f^{-1}) \tag{22}$$

$$(!SPINE f)^{-1} = (!ENIPS f^{-1}) \tag{23}$$

$$(!RIBS f)^{-1} = (!RIBS f^{-1}) \quad (24)$$

$$(!DEEPEN f)^{-1} = (!NEPEED f^{-1}) \quad (25)$$

We prove only the first case, i.e. that $bar \equiv (!FORK g)$ and $rab \equiv (!KROF g^{-1})$ are inverses of each other. Obviously this is true when we limit our attention to the set consisting only of the empty list $()$, which provides a base case for the inductive proof. For larger S-expressions $bar \circ rab$ is equivalent to

$$(bar(car\ side) . bar(cdr\ side)) \circ g \circ g^{-1} \circ (rab(car\ side) . rab(cdr\ side)) \quad (26)$$

In the middle $g \circ g^{-1}$ cancels out, and assuming the induction hypothesis is true, i.e. that $bar(rab(s)) = s$ for all smaller S-expressions s , it can be seen that the above produces an identity function for all larger S-expressions as well.

The cases involving other recursion schemes are proved similarly.

9. Implementing Recursion Schemata as Folds, Implications of Folds

Most recursion schemes listed in the previous section, can be implemented as *folds*, the concept described for example in [Hutton 1999] or [Gibbons 2005]:

Definition 30. Function $fold : \mathbb{F} \times \mathbb{T} \times \mathbb{S} \rightarrow \mathbb{T}$ for *lists* is defined as

$$fold(m, c, s) = \begin{cases} c & \text{if } s = () \\ m(x, fold(m, c, y)) & \text{if } s = cons(x, y) \end{cases} \quad (27)$$

where $c \in \mathbb{T}$, the set \mathbb{F} denotes the set of functions

$$m : \mathbb{S} \times \mathbb{T} \rightarrow \mathbb{T},$$

\mathbb{S} is the set of *S-expression*'s and \mathbb{T} is the range set for both m and $fold$ built upon it.

Similarly, function $foldubt : \mathbb{F} \times \mathbb{T} \times \mathbb{S} \rightarrow \mathbb{T}$ for *unlabeled binary trees* is defined as

$$foldubt(m, c, s) = \begin{cases} c & \text{if } s = \cdot \\ m(foldubt(m, c, a), foldubt(m, c, b)) & \text{if } s = \begin{array}{c} \color{red}{A} \swarrow \color{blue}{B} \\ \blacktriangledown \end{array} \end{cases} \quad (28)$$

Remark. In the context of this paper, where *unlabeled binary trees* are implemented in the form of *Nihilistic S-expression*'s, and where we restrict our attention to such *lists* which can be also realized as *Nihilistic S-expression*'s, $foldubt$ can always be implemented in terms of $fold$, provided its argument function f can recursively refer to the function being defined with $fold$.

We give three important properties of folds.

Lemma 2. *When is a function a fold?* [Gibbons, Hutton and Altenkirch 2001] If function F satisfies the implication

$$F(s) = F(t) \Rightarrow F(cons(a, s)) = F(cons(a, t)) \quad (29)$$

then function F can be implemented as $fold$ for *lists*.

If function F satisfies the implication

$$F(a) = F(b) \wedge F(s) = F(t) \Rightarrow F(\text{cons}(a, s)) = F(\text{cons}(b, t)) \quad (30)$$

then function F can be implemented as *foldubt* for *unlabeled binary trees*.

Remark. It should be noted that any function satisfying condition (30) satisfies also condition (29).

Lemma 3. *Universal Property.* ([Bird 1989], [Meertens 1983] or [Hutton 1999]). For any such function implementable as *fold* or *foldubt* there is a unique solution for function m in (27) and in (28) and the associated constant c . [XXX – CHECK!]

Lemma 4. *Fusion Property.* ([Bird 1989], [Meertens 1983], [Malcolm 1990], [Hutton 1999]). This can be stated as an implication

$$h(w(x, y)) = m(x, h(y)) \Rightarrow h \circ \text{fold}(w, c, s) = \text{fold}(m, h(c), s) \quad (31)$$

giving the condition which an arbitrary function h and functions m and w must satisfy that the composition of h and $\text{fold}(w, c, s)$ could be implemented as a fold of function m .

In Scheme, $\text{fold}(m, c, s)$ is implemented with the function (`fold-right m c s`). The higher order functions implementing the recursion schemes KROF, ENIPS, RIBS, DEEPEN and NEPEED for *constructively implemented* automorphisms are implemented in the following way as folds in Scheme:

```
(define (KROF g)
  (letrec ((h (lambda (s) (fold-right (lambda (x y) (g (cons (h x) y))) '() s)))) h))
  (32)
```

```
(define (ENIPS g) (lambda (s) (fold-right (lambda (x y) (g (cons x y))) '() s)))
  (33)
```

```
(define (RIBS g) (lambda (s) (fold-right (lambda (x y) (cons (g x) y)) '() s)))
  (34)
```

```
(define (DEEPEN g)
  (letrec ((h (lambda (s) (fold-right (lambda (x y) (cons (h x) y)) '() (g s))))) h))
  (35)
```

```
(define (NEPEED g)
  (letrec ((h (lambda (s) (g (fold-right (lambda (x y) (cons (h x) y)) '() s))))) h))
  (36)
```

9.1. Implications of properties of fold

Lemma 5. All Catalan automorphisms can be realized as folds, i.e. are *catamorphisms*.

Proof. Follows from the injectivity of automorphisms and from lemma 2.

Lemma 6. Except for RIBS, all the above mentioned recursion schemes have well-defined inverse operations.

Proof. The inverse operations for each of (ENIPS f), (SPINE f), (KROF f), (FORK f), (NEPEED f) and (DEEPEN f) can be realized as: (here f_{car} and f_{cdr} are shorthands for functions $(\lambda (s) (\text{cons } (f (\text{car } s)) (\text{cdr } s)))$ and $(\lambda (s) (\text{cons } (\text{car } s) (f (\text{cdr } s))))$ respectively).

$$(ENIPS^{-1} f) = f \circ f_{cdr}^{-1} \quad (37)$$

$$(SPINE^{-1} f) = f_{cdr}^{-1} \circ f \quad (38)$$

$$(KROF^{-1} f) = f \circ f_{car}^{-1} \circ f_{cdr}^{-1} \quad (39)$$

$$(FORK^{-1} f) = f_{car}^{-1} \circ f_{cdr}^{-1} \circ f \quad (40)$$

$$(NEPEED^{-1} f) = f \circ RIBS(f^{-1}) \quad (41)$$

$$(DEEPEN^{-1} f) = RIBS(f^{-1}) \circ f \quad (42)$$

It is easily seen and proven by induction that by applying the corresponding recursion scheme to these forms recovers the original automorphism f . Lemma 3 guarantees that these are indeed unique solutions.

Remark. The six recursion schemes *SPINE*, *ENIPS*, *FORK*, *KROF*, *DEEPEN* and *NEPEED* are thus bijective operations on the set of all Catalan automorphisms. However, in general they do not satisfy the group homomorphism condition, except in some special cases.

Lemma 7. Using the notation introduced above, we can define *RIBS* in terms of *SPINE* or *ENIPS*:

$$(RIBS f) = (SPINE f_{car}) = (ENIPS f_{car}) \quad (43)$$

Lemma 8. *KROF* is a composition of *NEPEED* and *ENIPS*:

$$(KROF f) = (NEPEED (ENIPS f)) \quad (44)$$

Proof. This is easiest to prove formally by considering the inverses of these operations. We have:

$$\begin{aligned} (ENIPS^{-1} (NEPEED^{-1} f)) &= (ENIPS^{-1} (f \circ RIBS(f^{-1}))) \\ &= f \circ RIBS(f^{-1}) \circ (f \circ RIBS(f^{-1}))_{cdr}^{-1} \\ &= f \circ RIBS(f^{-1}) \circ RIBS(f)_{cdr} \circ f_{cdr}^{-1} \\ &= f \circ f_{car}^{-1} \circ f_{cdr}^{-1} \\ &= (KROF^{-1} f) \end{aligned} \quad (45)$$

Lemma 9. *FORK* is a composition of *DEEPEN* and *SPINE*:

$$(FORK f) = (DEEPEN (SPINE f)) \quad (46)$$

Proof. The proof is similar to the one given for lemma 8.

Lemma 10. If we take an automorphism f and ENIPS -transform of another automorphism g , and consider their composition as an ENIPS -transform of some automorphism h :

$$f \circ (ENIPS g) = (ENIPS h) \quad (47)$$

then

$$h = f \circ g \circ f_{cdr}^{-1} \quad (48)$$

Proof. This can be also proved by considering the inverses. We have:

$$\begin{aligned} (ENIPS^{-1} (f \circ (ENIPS g))) &= (f \circ (ENIPS g)) \circ (f \circ (ENIPS g))_{cdr}^{-1} \\ &= f \circ (ENIPS g) \circ (ENIPS g)_{cdr}^{-1} \circ f_{cdr}^{-1} \\ &= f \circ g \circ f_{cdr}^{-1} \\ &= h \end{aligned} \quad (49)$$

Corollary. If automorphisms f and g are *non-recursive* Catalan automorphisms (i.e. $f, g \in A089840$), then so is also automorphism h .

Lemma 11. If we take an automorphism f and KROF -transform of another automorphism g , and consider their composition as a KROF -transform of some automorphism h :

$$f \circ (KROF g) = (KROF h) \quad (50)$$

then

$$h = f \circ g \circ f_{car}^{-1} \circ f_{cdr}^{-1} \quad (51)$$

Proof. The proof is similar to the one given for lemma 10. *Corollary.* If automorphisms f and g are *non-recursive* Catalan automorphisms (i.e. $f, g \in A089840$), then so is also automorphism h .

Lemma 12. If we have $g = (ENIPS f)$, we obtain an automorphism g_{cdr} as

$$g_{cdr} = (ENIPS f_{cdr}) \quad (52)$$

Proof. We substitute f for g and f^{-1} for f in the formula (48), as $g_{cdr} = f^{-1} \circ (ENIPS f)$.

Definition 31. We say that a Catalan automorphism f is *X-invariant* or *preserves X* if for all s ,

$$X(f(s)) = X(s) \quad (53)$$

Here X is a function whose domain is the same set of Catalan structures as on which automorphism f itself has been defined.

Remark. It is immediately seen that the identity automorphism **A001477* preserves all functions X defined on Catalan structures, and that the inverses and any composition of X -invariant automorphisms are also X -invariant.

Lemma 13. *If function X can be implemented as a fold, i.e. satisfies (30) and automorphism f is X -invariant, then f 's recursive derivations (FORK f), (KROF f), (SPINE f), (ENIPS f), (RIBS f), (DEEPEN f) and (NEPEED f) are X -invariant as well.*

Proof. We prove this first for the case (ENIPS f). By defining w as (define (w x y) (f (cons x y))) we see that

$$\begin{aligned}
X(w(x, y)) &= X(f(\text{cons}(x, y))) && \{ \text{By } w\text{'s definition} \} \\
&= X(\text{cons}(x, y)) && \{ f \text{ itself } X\text{-preserving} \} \\
&= \text{fold}(k, v, \text{cons}(x, y)) && \{ X \text{ is a fold, } v = (X ()), k \text{ unique} \} \\
&= k(x, \text{fold}(k, v, y)) && \{ \text{Expanding by one recursion step} \} \\
&= k(x, X(y)) && \{ \text{And then substituting } X \text{ back} \}
\end{aligned} \tag{54}$$

By substituting X for h , $()$ for c , w for w and k for m in (31), we see that the left side of the implication is satisfied, thus we are left with the right side of its implication, that is

$$\begin{aligned}
X \circ \text{fold}(w, c, s) &= \text{fold}(k, (X ()), s) \\
&= X(s)
\end{aligned} \tag{55}$$

that is, if $g = (\text{ENIPS } f)$, then

$$X(g(s)) = X(s) \tag{56}$$

which completes the proof that (ENIPS f) is also X -preserving, if automorphism f itself is.

Corollary. If f and (ENIPS f) are X -invariant, then so is (SPINE f), because the inverse of the latter form, (ENIPS f^{-1}) is X -invariant as well, because f^{-1} is also X -invariant.

Examples.

Definition 32. Function `length` in LISP and SCHEME returns the top-level length of the list, or, in the context of general trees (e), returns the degree of the root node, i.e. the first number of the associated Lukasiewicz-word. The associated OEIS-sequence is A057515.

Because `length` can be implemented as fold, like

$$(\text{define (length s) (fold-right (lambda (elem sum) (F (length elem) sum)) 0 s)) \tag{57}$$

where F has been defined as

$$(\text{define (F elem len prevlensum) (+ 1 prevlensum)) \tag{58}$$

it follows that if automorphism f preserves the top-level length of the list, then all the above mentioned recursive derivations of automorphism f also preserve the same property.

Definition 33. *Matula-Goebel encoding* is a bijection between unlabeled, non-oriented rooted general trees and natural numbers, discovered independently by [Matula 1968] and [Goebel 1980].

Definition 34. *Matula-Goebel signature* for unlabeled rooted plane general trees (e) assigns a unique natural number to each equivalence class of non-oriented rooted general trees that underlies each oriented tree. The associated OEIS-sequence is A127301, which can be computed as fold:

$$\text{(define (*A127301 s) (fold-right (lambda (t m) (* (A000040 (*A127301 t)) m)) 1 s))} \quad (59)$$

Definition 35. We say that automorphism g preserves non-oriented form of rooted general trees, if its signature permutation satisfies

$$A127301(sp_g(n)) = A127301(n) \quad (60)$$

for all $n \geq 0$.

Remark. Because A127301 can be computed as fold, it follows that if automorphism f preserves the Matula-Goebel signature of a general tree (i.e., its nonoriented form), then all the above mentioned recursive derivations of automorphism f also preserve the same property.

Furthermore, we have the following implication:

Proposition 2.

$$\begin{aligned} A127301(sp_g(n)) = A127301(n) \text{ for all } n \geq 0 \\ \rightarrow \\ A129593(sp_g(n)) = A129593(n) \text{ for all } n \geq 0. \end{aligned} \quad (61)$$

i.e. condition (60) implies condition (6).

Proof. It should be clear that preserving of the non-oriented form of a rooted plane (general) tree induces only such permutations on the corresponding Lukasiewicz-word, that although the degrees of each child-vertex might be permuted amongst themselves, they still always stay under their respective parent. Moreover, this in turn implies that each vertex stays at the same distance (or "level") from the root as it was.

Remark. The reverse condition is not true, see automorphism **A072797* for a counter-example.

Definition 36. *Matula-Goebel signature* for unlabeled rooted plane binary trees (c/d) assigns a unique natural number to each equivalence class of non-oriented rooted binary trees that underlies each plane binary tree. *XXX --- Wording... When we discard the orientation of a plane tree. etc.*

Because Matula-Goebel signature for binary trees can be computed with right fold, as

$$\text{(define (*A127302 s) (fold-right (lambda (t m) (* (A000040 (*A127302 t)) (A000040 m))) 1 s))} \quad (62)$$

it follows that if automorphism f preserves the Matula-Goebel signature of a binary tree (i.e., its nonoriented form), then all the above mentioned recursive derivations of automorphism f also preserve the same property.

Remark. As **A127302* = **A127301* \circ **A057123*, and also the latter has been defined as a fold:

$$\text{(define (*A057123 s) (fold-right (lambda (x y) (list (*A057123 x) y)) '() s))} \quad (63)$$

it implies that **A127302* can also be defined as a fold.

Remark. There are examples of functions whose invariancy is preserved by *certain* recursive derivations, although they might not be computable (XXX — ???) as folds, because they do not satisfy condition (29). Two examples follow.

If a Catalan automorphism f preserves the characteristic function of A072795 (*or more precisely, its signature-permutation does*), i.e. if

$$\chi_{A072795}(f(s)) = \chi_{A072795}(s) \quad (64)$$

holds for all s , then the property (8) holds for that automorphism. Incidentally, this can be defined also as a fold:

$$\text{(define (*char_A072795 s) (fold-right (lambda (x y) (if (null? x) 1 0)) 0 s))} \quad (65)$$

although, because the function does not satisfy the condition (30), it cannot be computed as a fold for binary trees.⁵

If a Catalan automorphism f preserves the characteristic function of A057548, i.e. if

$$\chi_{A057548}(f(s)) = \chi_{A057548}(s) \quad (66)$$

holds for all s , then the property (9) holds for that automorphism. Note that this function satisfies neither condition (29) nor (30).

Proposition 3. If any non-identity automorphism f satisfies the condition (8), that is, if it preserves $\chi_{A072795}$, that is, satisfies the condition (64), then the recursive derivations (*FORK* f), (*KROF* f), (*SPINE* f) or (*ENIPS* f) satisfy the stronger condition (10) only if

$$(f (\text{cons } ' () s)) = (\text{cons } ' () s). \quad (67)$$

Proof. Let's consider case $g = (\text{SPINE } f)$ first:

$$\begin{aligned} & \text{(define (g s)} \\ & \quad \text{(cond ((null? s) s)} \\ & \quad \quad \text{(else} \\ & \quad \quad \quad \text{(let ((t (f s)))} \\ & \quad \quad \quad \quad \text{(cons (car t) (g (cdr t)))} \\ & \quad \quad \quad \quad \text{)} \\ & \quad \quad \quad \text{)} \\ & \quad \text{)} \\ & \text{)} \end{aligned} \quad (68)$$

Now, if the condition (8) holds, that is, $(f (\text{cons } ' () s)) = (\text{cons } ' () (h s))$ for some h (it could be equal or different to f), then we know that

$$(g (\text{cons } ' () s)) = (\text{cons } ' () (g (h s))) \quad (69)$$

thus if f satisfies the condition (8) then its SPINE-derivative g will also satisfy the condition (8), but not the stronger condition (10), unless h is the identity (for example, when $f = *A089854$, $*A072797$, $*A089855$, $*A089856$ or $*A089857$, i.e. A089840[7]–A089840[11]). However, in that case f itself cannot satisfy condition (10) unless it is also identity.

⁵This is true only if we require $\chi_{A072795}$ to be strictly two-valued function, returning always either 0 or 1. However, if we create instead a variant function which returns a third distinct value just for $()$, then there's no problem.

Then, if we consider the case $g = (FORK f)$:

```
(define (g s)
  (cond ((null? s) s)
        (else
         (let ((t (f s)))
           (cons (g (car t)) (g (cdr t))))
         )
        )
  )
)
```

(70)

the conclusion is exactly same, as $(g ()) = ()$.

For cases $g = (ENIPS f)$ and $g = (KROF f)$ it is enough to refer (23) and (22) and to note that if any f satisfies the condition (8), then its inverse f^{-1} will also satisfy it.

Proposition 4. If automorphism f satisfies the condition (9), that is, if it preserves $\chi_{A057548}$, that is, satisfies the condition (66), then $(FORK f)$, $(KROF f)$, $(DEEPEN f)$ and $(NEPEED f)$ satisfy the stronger condition (12) only if

$$(f (\text{cons } s \ ' ())) = (\text{cons } s \ ' ()). \quad (71)$$

Proof. Let's consider case $g = (FORK f)$ first:

```
(define (g s)
  (cond ((null? s) s)
        (else
         (let ((t (f s)))
           (cons (g (car t)) (g (cdr t))))
         )
        )
  )
)
```

(72)

Now, if the condition (9) holds for f , that is, $(f (\text{cons } s \ ' ())) = (\text{cons } (h s) \ ' ())$ for some h (it could be equal or different to f), then we know that

$$(g (\text{cons } s \ ' ())) = (\text{cons } (g (h s)) \ ' ()) \quad (73)$$

thus if f satisfies the condition (9) then its SPINE-derivative g will also satisfy the condition (9), but not the stronger condition (12), unless h is the identity (for example, when $f = *A072796$, $*A089850$, $*A089851$, $*A089852$ or $*A089853$, i.e. A089840[2]–A089840[6]). However, in that case f itself cannot satisfy condition (12) unless it is also identity.⁶

Then, if we write the case $g = (DEEPEN f)$ in slightly different form, highlighting the

⁶These results have improved a lot since December 31 2007 revision of this paper when I made almost a diametrically opposite claims!

recursion to car-branch:

```

(define (g s)
  (cond ((null? s) s)
        (else
         (let ((t (f s)))
           (cons (g (car t)) (map g (cdr t))))
         )
        )
)

```

(74)

the conclusion is same as for *FORK*, as $(\text{map } g ()) = ()$.

For cases $g = (\text{KROF } f)$ and $g = (\text{NEPEED } f)$ it is enough to refer (22) and (25) and to note that if any f satisfies the condition (9), then its inverse f^{-1} will also satisfy it.

Remark. Either one of the conditions (10) and (12) imply self-embedding in scale $n : n + 1$, and that in turn implies that the cycle-count sequence of a bijection is genuinely monotone. Thus we see for example that all cycle-count sequences of Catalan bijections in range 2–11 in tables A122201 and A122202 are genuinely monotone.

Proposition 5. *All automorphisms obtained with RIBS are horizontally telescoping, that is, they satisfy the property defined in (10), regardless of the properties of the automorphism to which RIBS is applied to.*

Proof. Obvious. All automorphisms fix $()$.

Remark. Alternatively, we can consider this as a special case of SPINE, as $g = (\text{SPINE } f_{\text{car}})$ (see (43)):

```

(define (g s)
  (cond ((null? s) s)
        (else (cons (f (car t)) (g (cdr t))))
        )
)

```

(75)

In other words, if automorphism f satisfies the condition (9), that is, if it preserves $\chi_{A057548}$, then $(\text{SPINE } f)$ and $(\text{ENIPS } f)$ satisfy the condition (10).

Again, either one of the conditions (10) and (12) imply self-embedding in scale $n : n + 1$, and that in turn implies that the cycle-count sequence of a bijection is genuinely monotone. Thus we see for example that all cycle-count sequences of Catalan bijections in range 2–11 in tables A122203 and A122204 are genuinely monotone. (Ones in range 2–6 by proposition 4, and ones in range 7–11 by this remark.)

Proposition 6. *If automorphism f has a sequence F_f as its counts of fixed points sequence then automorphism $g \equiv (\text{RIBS } f)$ has as its counts of fixed points sequence*

$$F_g = \text{INVERT}(\text{RIGHT}(F_f)).$$

Here the operator RIGHT increases the indices of the sequence F_f by one (from zero to one-based sequence), and the operation INVERT is the one which Cameron calls the operator A in [Cameron 1985] and which appears with the name INVERT in [Bernstein and Sloane 1995]. It can be defined as the correspondence between two power series:

$$1 + \sum_{n=1}^{\infty} b_n x^n = \frac{1}{1 - \sum_{n=1}^{\infty} a_n x^n}$$

in which case sequence $b = \text{INVERT}(a)$. One interpretation is that b_n is the number of ordered arrangements of items of total weight n that can be formed if we have a_i types of items of weight i , $i \geq 1$.

Proof. We see that automorphism g fixes precisely those S-expressions whose “car ribs” are fixed by automorphism f . Thus in this case b_n gives the number of S-expressions of n nodes that are fixed by automorphism g and a_i gives all the S-expressions of $i - 1$ nodes that are fixed by automorphism f . Because in the “cdr spine” of the whole S-expression there is one extra node for each car-branch, the resulting numbers match. [DRAW A PICTURE!]

Proposition 7. *If automorphism g is a Lukasiewicz-word permuting, and automorphism f is the one that naturally embeds into it in scale $n : 2n$ as explained in lemma 1, then (FORK f) embeds in the same manner to (DEEPEN g).*

Proof. Follows from the properties of FORK and DEEPEN transforms.

10. Non-recursive bijections of binary trees and their recursive derivations

11. Catalan bijections in detail

11.1. automorphism *A069770

Fixed points counted by:	<i>AERATED</i> (A000108).
Cycles counted by:	A007595.
Max. cycle lengths given by:	<i>LEFT</i> (A046698).
LCM’s of cycle lengths given by:	<i>LEFT</i> (A046698).
Lukasiewicz-word permuting:	No.
Telescoping:	No.
Recursive compositions:	*A069770 = (FORK *A129604) = (KROF *A129604) = (ENIPS *A089859) = (SPINE *A089863). 7
Constructive definition:	(define (*A069770 s) (if (null? s) s (cons (cdr s) (car s))))
Destructive definition:	(define (*A069770! s) (if (pair? s) (let ((org-car (car s)) (set-car! s (cdr s)) (set-cdr! s org-car))))

⁷ Here *AERATED*(Axxxxxx) is used to indicate the OEIS-sequence Axxxxxx interpolated with zeros, and *LEFT*(Axxxxxx) (or *RIGHT*(Axxxxxx)) means the sequence Axxxxxx shifted left (or respectively, right) by one. Especially, *LEFT*(A046698) = 1,1,2,2,2,2,2,... is used to indicate the sequences of maximum cycle lengths and LCM’s of cycle lengths for involutions, and *LEFT*(A019590) = 1,1,0,0,0,0,... for the fixed point count sequences of those automorphisms that fix no structures which are larger than the trivial null and one-node structures.

This involution, which is the simplest Catalan automorphism after the identity automorphism, swaps the left and right-hand subtree of a binary tree. How it will partition the appropriate Catalan structures of size $n=3$ into three disjoint cycles is shown below:

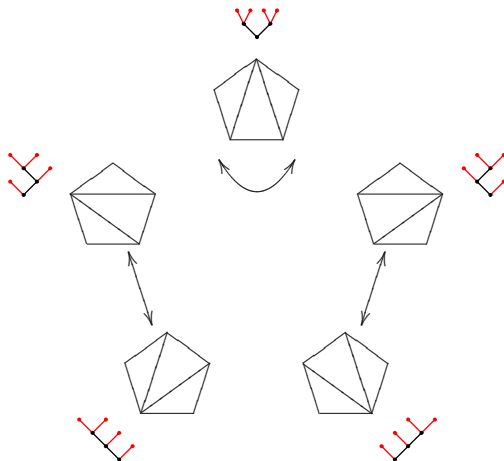


Figure 2: How automorphism $*A069770$ acts on binary trees of 3 internal nodes, and on the corresponding Eulerian polygon triangulations.

This automorphism fixes only binary trees whose left and right-hand subtree are identical (one extra, connecting node is located at the root), thus the number of fixed points in sub-range $[A014137(n-1)..A014138(n-1)]$ is given by the Catalan numbers interpolated with zeros:

$$f_{CS_{A069770}}(n) = \begin{cases} 0 & \text{if } n \text{ is even,} \\ C(\frac{n-1}{2}) & \text{if } n \text{ is odd.} \end{cases} \quad (76)$$

yielding

$$1, 1, 0, 1, 0, 2, 0, 5, 0, 14, 0, 42, 0, \dots$$

As with all involutions, the cycle-counts are obtained simply by taking the mean of Catalan numbers and the number of fixed points:

$$cc_{A069770}(n) = \begin{cases} \frac{C(n)}{2} & \text{if } n \text{ is even,} \\ \frac{C(n)+C(\frac{n-1}{2})}{2} & \text{if } n \text{ is odd.} \end{cases} \quad (77)$$

yielding

$$1, 1, 1, 3, 7, 22, 66, 217, 715, 2438, 8398, 29414, 104006, 371516, 1337220, \dots$$

This is the sequence A007595 in OEIS. Other manifestations/meanings given to it include *Number of necklaces of 2 colors with 2n beads and n-1 black ones* [Meeussen, 2002] and *Number of even permutations avoiding 132*.

Proposition 8. The following two identities hold:

$$(FORK (*A069770 \circ g)) = (FORK *A069770) \circ (FORK g) \quad (78)$$

$$(KROF (g \circ *A069770)) = (KROF g) \circ (KROF *A069770) \quad (79)$$

Proof. We prove (79) by considering the inverse of *KROF*-operation. We let X stand for the right side of (79), and so we have:

$$\begin{aligned}
(KROF^{-1} X) &= (KROF g) \circ (KROF *A069770) \\
&\circ ((KROF g) \circ *A057163)_{car}^{-1} \\
&\circ ((KROF g) \circ *A057163)_{cdr}^{-1} \\
&= (KROF g) \circ *A057163 \circ *A057163_{car} \circ (KROF g)_{car}^{-1} \circ *A057163_{cdr} \circ (KROF g)_{cdr}^{-1} \\
&= (KROF g) \circ *A057163 \circ *A057163_{car} \circ *A057163_{cdr} \circ (KROF g)_{car}^{-1} \circ (KROF g)_{cdr}^{-1} \\
&= (KROF g) \circ *A069770 \circ (KROF g)_{car}^{-1} \circ (KROF g)_{cdr}^{-1} \\
&= (KROF g) \circ (KROF g)_{cdr}^{-1} \circ *A069770 \circ (KROF g)_{cdr}^{-1} \\
&= (KROF g) \circ (KROF g)_{cdr}^{-1} \circ (KROF g)_{car}^{-1} \circ *A069770 \\
&= g \circ *A069770
\end{aligned} \quad (80)$$

The case (79) is proved similarly.

Proposition 9. The following two identities hold:

$$(FORK (*A057163 \circ g \circ *A057163)) = *A057163 \circ (FORK g) \circ *A057163 \quad (81)$$

$$(KROF (*A057163 \circ g \circ *A057163)) = *A057163 \circ (KROF g) \circ *A057163 \quad (82)$$

Proof. This is easily proved graphically.

11.2. Automorphism $*A057163$

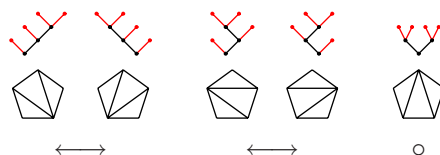
Fixed points counted by:	<code>AERATED(A000108).</code>
Cycles counted by:	<code>A007595.</code>
Max. cycle lengths given by:	<code>LEFT(A046698).</code>
LCM's of cycle lengths given by:	<code>LEFT(A046698).</code>
Lukasiewicz-word permuting:	No.
Telescoping:	No.
Recursive composition:	<code>*A057163 = (FORK *A069770).</code> <code>(define (*A057163 s)</code> <code> (cond ((not (pair? s)) s)</code> <code> (else (cons (*A057163 (cdr s))</code> <code> (*A057163 (car s))))</code>
Constructive definition:	<code>)</code> <code>)</code> <code>)</code> <code>(define (*A057163! s)</code> <code> (cond ((pair? s)</code> <code> (*A069770! s)</code> <code> (*A057163! (car s))</code> <code> (*A057163! (cdr s))</code>
Destructive definition:	<code>)</code> <code>)</code> <code>s</code> <code>)</code>

11.3. Automorphism **A057163*

Fixed points counted by: *AERATED*(A000108).
 Cycles counted by: A007595.
 Max. cycle lengths given by: *LEFT*(A046698).
 LCM's of cycle lengths given by: *LEFT*(A046698).
 Lukasiewicz-word permuting: No.
 Telescoping: No.
 Recursive compositions: **A057163* = (FORK **A69770*)
 (define (**a057163* s)
 Constructive variant: (cond ((null? s) s)
 (else (cons (**a057163* (cdr s)) (**a057163* (car s))))))
 (define (**a057163!* s)
 Destructive variant: (cond ((pair? s) (swap! s) (**a057163!* (car s)) (**a057163!* (cdr
 s))

Reflect binary trees and polygon triangulations.

This involution reflects the binary trees and polygon triangulations. It is obtained by applying automorphism **A069770* recursively down to every branch of a binary tree, and it has the same cycle-count, fixed point count, max. and LCM-sequences as the former. However, the fixed points themselves generally are not the same, as this one fixes the *symmetric* binary trees, i.e. ones whose left and right-hand sides are (usually) not identical, but instead *mirror images* of each other. How this automorphism will partition the appropriate Catalan structures of size $n=3$ into three disjoint cycles is shown below:



11.4. Automorphism **A072796*

Fixed points counted by: A073190.
 Cycles counted by: A073191.
 Max. cycle lengths given by: A046698.
 LCM's of cycle lengths given by: A046698.
 Lukasiewicz-word permuting: No.
 Telescoping: No.
 Recursive compositions: (define (**a072796* s)
 Constructive variant: (cond ((null? s) s)
 ((not (pair? (cdr s))) s)
 (else (cons (cadr s) (cons (car s) (caddr s))))))
 (define (**a072796!* s)
 Destructive variant: (cond ((null? s) s)
 ((not (pair? (cdr s))) s)
 (else (swap! s) (robr! s) (swap! (cdr s)) s)))

Exchange the two leftmost branches of general trees if the degree of root larger than 1, otherwise keep the tree intact.

This non-recursive automorphism exchanges the two leftmost branches of general trees if the degree of tree's root is > 1 , otherwise it keeps the tree intact. Thus the fix point count sequence

gives the number of general plane trees which are either empty (the case $n = 0$), or whose root degree is either 1 (i.e. the planted trees) or the two leftmost subtrees (of the root node) are identical. This can be computed as a sum of planted trees (the first term) and a convolution, where the left hand side of product gives the number of trees that can occur as the two identical leftmost subtrees, and the right hand side of the product gives the number of possibilities for the rest of tree.

$$A073190(n) = C(n-1) + \sum_{\substack{i=0 \\ 2|(n-i)}}^{n-2} C\left(\frac{n-i-2}{2}\right) C(i) \quad (83)$$

yielding

$$1, 1, 2, 3, 8, 20, 60, 181, 584, 1916, 6476, 22210, 77416, \dots$$

Cycle counts follow as before:

$$A073191(n) = \frac{C(n) + A073190(n)}{2}. \quad (84)$$

yielding

$$1, 1, 2, 4, 11, 31, 96, 305, 1007, 3389, 11636, 40498, 142714, \dots$$

The automorphism $*A072797$ is obtained by conjugating $*A072796$ with $*A057163$, thus it has exactly same fixed point and cycle count sequences. It has two interesting properties:

First, although it does not satisfy condition (60) it still satisfies condition (6), i.e. is Lukasiewicz-permuting.

Also, it has at least two recursive derivations, namely

$$*A069775 = *A057163 \circ (SPINE *A057163 \circ *A072797 \circ *A057163) \circ *A057163 \quad (85)$$

and

$$*A069776 = *A057163 \circ (ENIPS *A057163 \circ *A072797 \circ *A057163) \circ *A057163 \quad (86)$$

that satisfy the same condition.

Furthermore, $*A072797$ induces it *itself* with the method described in (7), i.e.

$$*A072797 = *A083927 \circ *A072797 \circ *A057123 \quad (87)$$

and similarly, it seems that

$$*A069775 = *A083927 \circ *A069775 \circ *A057123 \quad (88)$$

and

$$*A069776 = *A083927 \circ *A069776 \circ *A057123 \quad (89)$$

XXX — The latter two remain to be proved.

From the proposition 7 then follows that $(FORK *A072797)$, (i.e. automorphism $*A082325$, which is the $*A057163$ -conjugate of $*A057511$), embeds in the same manner into $(DEEPEN *A072797) = *A122313$.

XXX- Automorphism equations...

For what automorphisms f and g , $(SPINE f \circ g) = (SPINE f) \circ (SPINE g)$? At least when f is one of the automorphisms that leave the right subtree of binary trees intact. Any others?

For what automorphisms f , $f = *A083927 \circ f \circ *A057123$? At least when $f = *A001477$, $*A072797$, $*A069775$, $*A069776$.

For what automorphisms f , $f^2 = *A083929 \circ f^3 \circ *A083930$? At least when $f = *A001477$, $*A057505$, $*A057506$.

Prove all these.

Also these:

$$*A057501 = *A083927 \circ *A085159 \circ *A057123$$

and

$$*A057502 = *A083927 \circ *A085160 \circ *A057123$$

11.5. Automorphism $*A057509/*A057510$

Fixed points counted by:	A034731.
Cycles counted by:	A003239.
Max. cycle lengths given by:	<i>RIGHT</i> (A028310).
LCM's of cycle lengths given by:	<i>RIGHT</i> (A003418).
Lukasiewicz-word permuting:	Yes, the restriction to binary trees induces $*A069770$.
Telescoping:	No.
Recursive compositions:	$*A057509 = (\text{SPINE } *A72796)$ $*A057510 = (\text{ENIPS } *A72796)$
Compositions:	$*A057509 = *A57501 \circ *A69770$ $*A057510 = *A69770 \circ *A57502$
Constructive variant for rotate left using Lisp/Scheme built-in function <i>append</i> :	<pre>(define (*a057509 s) (cond ((null? s) s) (else (append (cdr s) (list (car s))))))</pre>
Constructive, recursive variant for rotate left:	<pre>(define (*a057509v2 s) (cond ((null? s) s) ((null? (cdr s)) s) (else (cons (car (cdr s)) (*a057509v2 (cons (car s) (cdr s))))))</pre> <pre>(define (*a057509! s) (cond ((pair? s) (swap! s) (*a057509! s)) (s)))</pre>
Destructive variants, for both rotate left and right, composed of <i>swap!</i> and handshake rotates:	<pre>(define (*a057510! s) (cond ((pair? s) (*a057502! s) (swap! s)) (s)))</pre>

Shallow Rotate general trees and parenthesizations.

This automorphism rotates the top-level branches of general trees by one step. It fixes only the trees whose toplevel subtrees are all identical, and thus the number of fixed points can be computed as (one is subtracted from the divisor d of n , as for each subtree there is an extra node in the main toplevel stem of the tree.)

$$f_{CS_{A057509}}(n) = \begin{cases} 1 & \text{if } n \text{ is zero,} \\ \sum_{d|n} C(d-1) & \text{otherwise.} \end{cases} \quad (90)$$

yielding

1, 1, 2, 3, 7, 15, 46, 133, 436, 1433, 4878, 16797, 58837, 208013, 743034, ...

This is the sequence A034731 in OEIS, originally submitted by Erich Friedman with the name *Dirichlet convolution of $b_n = 1$ with Catalan numbers*.

The cycle counts are given by A003239, *number of rooted planar trees with n non-root nodes: circularly cycling the subtrees at the root gives equivalent trees*. This is computed as:

$$A003239(n) = \frac{1}{(2n)} \sum_{d|n} \phi(n/d) \binom{2d}{d} \quad (91)$$

yielding

$$1, 1, 2, 4, 10, 26, 80, 246, 810, 2704, 9252, 32066, 112720, 400024, \dots$$

Another manifestation/meaning given to it include *number of necklaces with $2n$ beads, n white and n black*, from which the above formula is easily seen to be derived from.

Still more manifestations mentioned are *number of terms in polynomial expression for permanent of generic circulant matrix of order n* and *number of equivalence classes of n -compositions of n under cyclic rotation*.

It should be easy to see that the least common multiples of cycle sizes for trees of size n is given by $a(n) = 1$ when $n = 0$ and for $n \geq 1$ by $a(n) = \text{LCM}_{i=1}^{n-1} i$, thus yielding the right shifted version of OEIS-sequence A003418: 1,1,1,2,6,12,60,60,420,840,2520,2520,27720,27720,360360,...

11.6. Automorphism **A057508*

Fixed points counted by: A073192.

Cycles counted by: A073193.

.

.

Lukasiewicz-word permuting: Yes, the restriction to binary trees induces **A069770*.

Telescoping: No.

Recursive compositions: **A057508* = (ENIPS **A57509*) = (SPINE **A57510*)

```
(define *a057508
  reverse)

(define (*a057508v2 s)
  (cond ((null? s) (list))
        (else (append (*a057508v2 (cdr s)) (list (car s))))))
```

```
(define (*a057508v3 s)
  (cond ((null? s) s)
        ((null? (cdr s)) s)
        (else
```

Constructive, deeply recursive variant. [See <http://www.research.att.com/cgi-bin/access.cgi/as/njas/sequences/eisA.cgi?Anum=A033538>].

```
(cons (car (*a057508v3 (cdr s)))
      (cons (car s) (*a057508v3 (cdr (*a057508v3 (cdr s))))))
```

```
(define (*a057508v4 a)
  (let loop ((a a) (b (list)))
    (cond ((not (pair? a)) b)
          (else (loop (cdr a) (cons (car a) b))))))

(define (*a057508! s)
  (cond ((pair? s) (*a057508! (cdr s)) (*a057509! s))
        (s)
```

Two destructive variants:

```
(define (*a057508v2! s)
  (cond ((pair? s) (*a057510! s) (*a057508v2! (cdr s))))
s)
```

Shallow Reverse general trees and parenthesizations.

This automorphism reverses the top-level branches of general trees. Thus the fix point count sequence gives the number of general plane trees which are palindromic in shallow sense, i.e. whose k -th toplevel subtree from the left is equal with the k -th subtree from the right, for all their subtrees. This can be computed as a convolution, where the right hand side of the product gives the number of different kind of trees that can occur in the middle of toplevel list (when the number of toplevel subtrees is odd), and the left hand side of product gives the number of possibilities that can occur at the left and right hand sides of an S-expression so that it will be symmetric.

$$A073192(n) = \sum_{\substack{i=0 \\ 2|(n-i)}}^n C\left(\frac{n-i}{2}\right) \check{C}(i-1) \quad (92)$$

where $\check{C}(n) = 1$ if $n = -1$ and otherwise as $C(n)$ (the n th Catalan number). This yields the sequence

1, 1, 2, 3, 8, 18, 54, 155, 500, 1614, 5456, 18630, 64960, ...

Cycle counts follow as before:

$$A073193(n) = \frac{C(n) + A073192(n)}{2}. \quad (93)$$

yielding

1, 1, 2, 4, 11, 30, 93, 292, 965, 3238, 11126, 38708, 136486, ...

11.7. Automorphism **A057164*

Fixed points counted by:	A001405.
Cycles counted by:	<i>LEFT</i> (A007123).
Max. cycle lengths given by:	A046698.
LCM's of cycle lengths given by:	A046698.
Lukasiewicz-word permuting:	Yes, the restriction to binary trees induces <i>*A057163</i> .
Telescoping:	No.
Recursive compositions:	<i>*A057164</i> = (FORK <i>*A57510</i>) = (KROF <i>*A57509</i>) = (DEEP <i>*A57508</i>) <pre>(define (*a057164 s) (cond ((null? s) s) ((null? (cdr s)) (cons (*a057164 (car s)) (list))) (else (append (*a057164 (cdr s)) (*a057164 (cons (car s) (list)))))) (define (*a057164v2 s) (cond ((null? s) s) ((null? (cdr s)) (list (*a057164v2 (car s)))) (else (cons (*a057164v2 (car (*a057164v2 (cdr s)))) (*a057164v2 (cons (car s) (*a057164v2 (cdr (*a057164v2 (cdr s))))))))) (define (*a057164! s) (cond ((pair? s) (*a057164! (car s)) (*a057164! (cdr s)) (*a057164! s)) (else s)))</pre>
Constructive, recursive variant using Lisp/Scheme built-in function append:	
Constructive, deeply recursive variant:	
Destructive variant:	

Deep Reverse general trees and parenthesizations.

The number of fixed points, i.e. the symmetric general trees and Dyck paths, is given by central binomial coefficients

$$fcs_{A057164}(n) = \binom{n}{\lfloor n/2 \rfloor} \quad (94)$$

1,1,2,3,6,10,20,35,70,126,252,462,924,1716,3432,6435,12870,... which is the sequence A001405 in OEIS. For the proof, see e.g. [].

The cycle counts is computed as for all involutions $cc_{A057164}(n) = \frac{C(n)+fcs_{A057164}(n)}{2}$, giving 1,1,2,4,10,26,76,232,750,2494,8524,29624,104468,... which is the sequence A007123 in OEIS. It has been submitted with the name *Number of connected unit interval graphs with n nodes; also bracelets (turn over necklaces) with n black beads and n-1 white beads*. This connection with binary bracelets can be easily seen/proved by considering Raney's lemma as explained by Graham, Knuth and Patashnik.

This table can be computed by defining $A079216(k, 0) = 1$, and then computing for the larger values of n with the following recurrence:

$$A079216(k, n) = \sum_{\substack{r=1 \\ (r/\gcd(r,k))|n}}^n \sum_{c_1+\dots+c_{\gcd(r,k)}=n/(r/\gcd(r,k))} \prod_{i=1}^{\gcd(r,k)} A079216(\text{lcm}(r, k), c_i - 1) \quad (95)$$

The outer sum ranges over all the values $r = 1..n$ the degree of the root node of an n -edge general tree may obtain, with the additional condition that $r/\gcd(r, k)$ (the cycle length) must divide n . Here the inner sum is iterated over each composition of $n/(r/\gcd(r, k))$ into $\gcd(r, k)$ (i.e., the number of cycles) parts, and the innermost product ranges over each of the composants. *XXX - Apply generatingfunctionology and other black magic here. E.g. if we take product over the composants, then isn't it related to INVERT-transform then?*

Now the count of fixed points can be computed as

$$fcs_{A057511}(n) = \begin{cases} 1 & \text{if } n \text{ is zero,} \\ \sum_{d|n} A079216(n/d, d-1) = A079216(1, n) & \text{otherwise.} \end{cases} \quad (96)$$

yielding

$$1, 1, 2, 3, 5, 6, 10, 11, 18, 21, 34, 35, 68, 69, 137, 148, 316, 317, 759, 760, \dots$$

This is the sequence [A057546](#) in OEIS. Note that $fcs_{A057511}(p) = fcs_{A057511}(p-1) + 1$ for all primes p .

The cycle count sequence follows then as

$$cc_{A057511}(n) = \begin{cases} 1 & \text{if } n \text{ is zero,} \\ \frac{1}{A003418(n-1)} \sum_{i=1}^{A003418(n-1)} A079216(i, n) & \text{otherwise.} \end{cases} \quad (97)$$

yielding

$$1, 1, 2, 4, 9, 21, 56, 153, 451, 1357, 4212, 13308, 42898, 140276, 465324, \dots$$

This is the sequence [A057513](#) in OEIS. Note that we might as well write the formula in the form

$$cc_{A057511}(n) = \frac{1}{n!} \sum_{i=1}^{n!} A079216(i, n) \quad (98)$$

for $n \geq 1$, as $n!$ is always a multiple of $A003418(n-1)$. *XXX - this might or might not help to reduce the formula to a more practical form. Explain also how it is derived!*

11.9. Automorphism **A069767/*A069768*

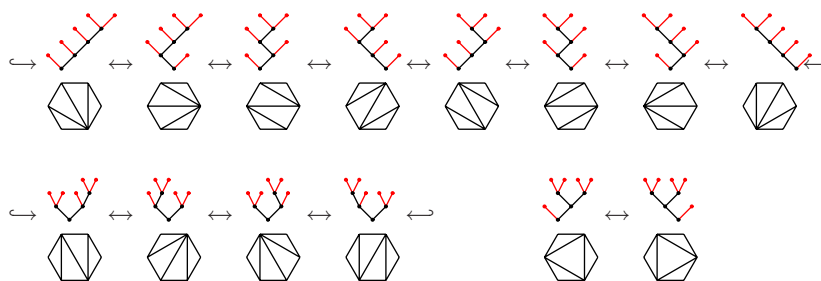
Fixed points counted by: A036987.
 Cycles counted by: A073431.
 Max. cycle lengths given by: A011782.
 LCM's of cycle lengths given by: A011782.
 Lukasiewicz-word permuting: No.
 Telescoping: No.
 Recursive compositions: **A069767* = (SPINE **A69770*)
**A069768* = (ENIPS **A69770*)

```
(define (*a069767! s)
  (cond ((pair? s) (swap! s) (*a069767! (cdr s))))
s)
(define (*a069768! s)
  (cond ((pair? s) (*a069768! (cdr s)) (swap! s))
s)
```

Destructive variants:

Swap recursively the other side of binary tree.

How it will partition the fourteen binary trees of four internal nodes (and corresponding polygon triangulations) into three disjoint cycles is shown below:



Remark. Any Catalan automorphism f for which $A127302(f(n)) = A127302(n)$ holds for all n , keeps the nonoriented form of the underlying binary tree intact. This holds also for $f = *A069767$ and $f = *A069768$. In each set of $\{C_n$ structures of size $n\}$, i.e. C_n binary trees of n internal (branching nodes), there is a subset of 2^{n-1} binary trees whose height (i.e. max depth) is equal to their size. The above condition implies that this subset is closed with respect to any automorphism that satisfies that condition.

Furthermore, automorphisms **A069767* and **A069768* act transitively on that subset, i.e. those trees form a single cycle of their own, as can be seen below.

If we map that subset to a binary strings of length $n - 1$, let the root node stand for the least significant bit, and the next-to-top node on those trees stand the most significant bit, and mark 0 when the next node upwards is at the right, and 1 when it is at left, we get the sequence of binary words (in this case, of three bits) shown below on the top of the eight binary trees belonging to that closed cycle. It is easy to see that these automorphisms induce the well-known binary wrap-around ("odometer") increment/decrement algorithm on the binary strings that are in bijective correspondence with such binary trees.

Proposition 10. The following two identities hold:

$$*A069767 = (KROF *A069768) \tag{99}$$

$$*A069768 = (FORK *A069767) \quad (100)$$

Proof. We prove (99) by considering the inverse of $KROF$ -operation. We have:

$$\begin{aligned}
(KROF^{-1} *A069767) &= *A069767 \circ *A069767_{car}^{-1} \circ *A069767_{cdr}^{-1} \\
&= (SPINE *A069770) \circ *A069768_{car} \circ *A069768_{cdr} \\
&= (SPINE *A069770)_{cdr} \circ *A069770 \circ *A069768_{car} \circ *A069768_{cdr} \\
&= *A069767_{cdr} \circ *A069768_{cdr} \circ *A069770 \circ *A069768_{cdr} \{XXX - EXPLAIN!\} \\
&= *A069770 \circ (ENIPS *A069770)_{cdr} \\
&= (ENIPS *A069770) \\
&= *A069768
\end{aligned} \quad (101)$$

The case (100) follows from the identity $(KROF f)^{-1} = (FORK f^{-1})$.

Proposition 11. The following two identities hold:

$$*A069768 = (KROF *A069767) \quad (102)$$

$$*A069767 = (FORK *A069768) \quad (103)$$

Proof. We prove (102) by considering the inverse of $KROF$ -operation. We have:

$$\begin{aligned}
(KROF^{-1} *A069768) &= *A069768 \circ *A069768_{car}^{-1} \circ *A069768_{cdr}^{-1} \\
&= (ENIPS *A069770) \circ *A069767_{car} \circ *A069767_{cdr} \\
&= (ENIPS *A069770) \circ *A069767_{cdr} \circ *A069767_{car} \\
&= *A069770 \circ (ENIPS *A069770)_{cdr} \circ *A069767_{cdr} \circ *A069767_{car} \\
&= *A069770 \circ (ENIPS *A069770)_{cdr} \circ (SPINE *A069770)_{cdr} \circ *A069767_{car} \\
&= *A069770 \circ *A069767_{car} \\
&= *A069770 \circ (SPINE *A069770)_{car} \\
&= (SPINE *A069770)_{cdr} \circ *A069770 \\
&= *A069767
\end{aligned} \quad (104)$$

The case (103) follows from the identity $(KROF f)^{-1} = (FORK f^{-1})$.

Digression 1. We have proved that $*A069767 = (FORK^2 *A069767) = (KROF^2 *A069767)$ and similarly $*A069768 = (FORK^2 *A069768) = (KROF^2 *A069768)$. A question remains: Are there any other nonidentity Catalan automorphisms, for which a repeated application of $FORK$ or some other automorphism of Catalan automorphisms would produce a cycle of finite length? Does the equation

$$(FORK^2 g) = g \quad (105)$$

have just two non-identity solutions?

One subset of cases which satisfy the condition (105) is when we have

$$(FORK g) = g^{-1} \quad (106)$$

in which case, by applying $FORK$ to both sides, we get

$$(FORK^2 g) = (FORK g^{-1}) = g \quad (107)$$

By inverting the both sides of (106) to get

$$(FORK\ g)^{-1} = g \quad (108)$$

and then applying (22) at the left side, we get

$$(KROF\ g^{-1}) = g \quad (109)$$

Then by applying (39), we get

$$g^{-1} = g \circ g_{car}^{-1} \circ g_{cdr}^{-1} \quad (110)$$

Inverting both sides, we get:

$$g = g_{cdr} \circ g_{car} \circ g^{-1} \quad (111)$$

and then multiplying from the right by g we obtain:

$$g^2 = g_{cdr} \circ g_{car} \quad (112)$$

which is equal to

$$g^2 = g_{car} \circ g_{cdr} \quad (113)$$

as the elements of the right hand side composition commute with each other.

It should be clear, that apart from the trivial identity bijection, and $*A069767$ and $*A069768$ themselves, all their higher powers satisfy this condition as well. For example, bijections $*A073290$ – $*A073299$. Thus, in the infinite cyclic group generated by $*A069767$ and its inverse $*A069768$, which is isomorphic to the additive group of integers, $(\mathbb{Z}, +)$, both $FORK$ and $KROF$ satisfy the group homomorphism condition, and specifically, they are automorphisms of that group, mapping each element to its own inverse.

If we apply (40) directly to (105) we first get

$$(FORK\ g) = g_{car}^{-1} \circ g_{cdr}^{-1} \circ g \quad (114)$$

and after the second application

$$g = (g_{car}^{-1} \circ g_{cdr}^{-1} \circ g)_{car}^{-1} \circ (g_{car}^{-1} \circ g_{cdr}^{-1} \circ g)_{cdr}^{-1} \circ (g_{car}^{-1} \circ g_{cdr}^{-1} \circ g) \quad (115)$$

which after some massaging gets to

$$g = (g^{-1} \circ g_{car} \circ g_{cdr})_{car} \circ (g^{-1} \circ g_{car} \circ g_{cdr})_{cdr} \circ (g_{car}^{-1} \circ g_{cdr}^{-1} \circ g) \quad (116)$$

and further to

$$g = g_{car}^{-1} \circ g_{caar} \circ g_{cdar} \circ g_{cdr}^{-1} \circ g_{cadr} \circ g_{cddr} \circ g_{car}^{-1} \circ g_{cdr}^{-1} \circ g \quad (117)$$

Then, multiplying from the right by g^{-1} , and letting those elements that can commute past each other to do so, in most appropriate fashion, we get

$$id = g_{car}^{-1} \circ g_{cdr}^{-1} \circ g_{caar} \circ g_{cdar} \circ g_{cadr} \circ g_{cddr} \circ g_{cdr}^{-1} \circ g_{car}^{-1} \quad (118)$$

Multiplying from the both sides by g_{car} we get

$$g_{car}^2 = g_{cdr}^{-1} \circ g_{caar} \circ g_{cdar} \circ g_{cadr} \circ g_{cddr} \circ g_{cdr}^{-1} \quad (119)$$

and then doing the same thing with g_{cdr} (note that g_{car} and g_{cdr} commute with each other):

$$g_{car}^2 \circ g_{cdr}^2 = g_{caar} \circ g_{cdar} \circ g_{cadr} \circ g_{cddr} \quad (120)$$

As the left and right hand sides are acted independently upon, with nothing crossing from one side to the other, this is equivalent that two different conditions are satisfied both at the same time:

$$g_{car}^2 = g_{caar} \circ g_{cdar} \tag{121}$$

and

$$g_{cdr}^2 = g_{cadr} \circ g_{cddr} \tag{122}$$

that is, this is equivalent that the condition (113) is applied both to the left and right hand side of the binary tree.

Are there any cases, where (121) and (122) would hold, but (113) itself would not hold?

Does the equation

$$(FORK^3 g) = g \tag{123}$$

admit non-identity solutions?

Fixed count and cycle count sequences. To compute the $f_{csA069767}$ and $cc_{A069767}$ we need a function that gives the number of rooted plane binary trees of size n and "contracted height" k .

This table has been submitted as A073346 into OEIS:

$A073346_{k,n} :=$	$k \setminus n$	0	1	2	3	4	5	6	7	...
	0	1	1	0	1	0	0	0	1	...
	1	0	0	2	0	2	2	0	0	...
	2	0	0	0	4	4	8	12	12	...
	3	0	0	0	0	8	16	40	80	...

This is a variant of table A073345, which was known to the Chinese mathematician Ming An-Tu, who gave the following recurrence in the 1730s: (give it here) [Luo Jian-Jin, 1995].

The count of fixed points, the row 0 of the above table, is given by Fredholm-Rueppel sequence. the sequence A036987 in OEIS. It is defined as a characteristic function of $2^n - 1$, giving the terms:

1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ...

The cycle count sequence follows then as

$$\begin{aligned}
 cc_{A069767}(n) &= \frac{1}{2^{n-1}} \sum_{i=1}^{2^{n-1}} \sum_{j=0}^{A007814(i)} A073346(j, n) \\
 &= -1 + \frac{1}{2^{n-2}} \sum_{i=1}^{2^{n-1}} A073346(A007814(i), n) \\
 &= \frac{1}{2^n} \sum_{i=0}^n 2^{n-i} A073346(i, n) \\
 &= \sum_{i=0}^n 2^{-i} A073346(i, n)
 \end{aligned} \tag{124}$$

Here we use sequence A007814, *Exponent of highest power of 2 dividing n*, i.e. the binary carry sequence,

0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 5, 0, 1, 0, 2, 0, ...

as a kind of “filter”, with which XXX ... are selected.

Because entries on the row k of table A073346 are always multiples of 2^k , they can be divided out, to form another table A074079, and the cycle count formula can be represented in terms of that table.

$$cc_{A069767}(n) = \sum_{i=0}^n A074079(i, n) \quad (125)$$

The formula gives the terms

1, 1, 1, 2, 3, 6, 12, 28, 65, 160, 408, 1074, 2898, 7998, 22508, 64426, 187251, ...

This is the sequence A073431 in OEIS.

XXX - Lots of explaining here! Also the cc-sequence can be probably reduced further, by using generatingfunctionological approach.

11.10. Automorphism *A057161/*A057162

Fixed points counted by: `LEFT(A019590).`
Cycles counted by: `LEFT(LEFT(A001683)).`
Max. cycle lengths given by: `A057544.`
LCM's of cycle lengths given by: `A057544.`
Lukasiewicz-word permuting: `No.`
Telescoping: `No.`
Compositions: `*A057161 = *A57508 o *A69767 = *A69767 o *A69769 = *A57163 o *A57163`
`*A057162 = *A69768 o *A57508 = *A69769 o *A69768 = *A57163 o *A57163`

```
(define (*a057161 a)
  (let loop ((a a) (b (list)))
    (cond ((not (pair? a)) b)
          (else (loop (car a) (cons (cdr a) b))))))
```

Constructive, tail-recursive variants:

```
(define (*a057162 a)
  (let loop ((a a) (b (list)))
    (cond ((not (pair? a)) b)
          (else (loop (cdr a) (cons b (car a))))))
(define (*a057161v2 s)
  (cond ((null? s) s)
        (else (append (*a057161v2 (car s)) (list (cdr s))))))
```

Constructive, recursive variant for A057161 using Lisp/Scheme built-in function append:

```
(define (*a057161! s)
  (*a069769! s)
  (*a069767! s)
  s)
```

Destructive variants, composed of other destructively implemented gatomorphisms:

```
(define (*a057162! s)
  (*a057508! s)
  (*a069768! s)
  s)
```

Rotate polygon triangulations.

The cycle count sequence is easiest to define in terms of A001683, *Number of one-sided triangulations of the disk*, i.e. the triangulations of Eulerian polygons with n sides.

$$A001683(n) = \frac{1}{n}C(n-2) + \frac{1}{2}c\left(\frac{n}{2}-1\right) + \frac{2}{3}c\left(\frac{n}{3}-1\right) \quad (126)$$

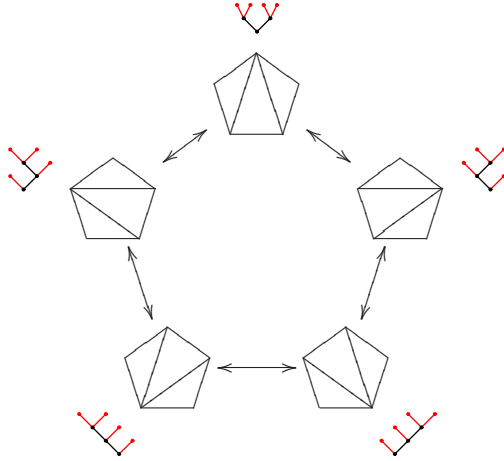


Figure 3: How automorphism $*A057161/A057162$ acts on binary trees of 3 internal nodes, and on the corresponding Eulerian polygon triangulations.

after which we can simply define

$$cc_{A057161}(n) = A001683(n + 2) \quad (127)$$

because a binary tree of n internal nodes corresponds to a triangulation of a polygon of $n + 2$ vertices. This gives the terms

$$1, 1, 1, 1, 4, 6, 19, 49, 150, 442, 1424, 4522, 14924, \dots$$

Connections with other bijections. We are interested how $*A057161$ and $*A057162$ can be represented as folds. Specifically, we want to solve the unique f from equation:

$$*A057161 = (SPINE f) \quad (128)$$

and unique g from equation:

$$*A057162 = (ENIPS g) \quad (129)$$

It is enough to solve only the other one, as then the other solution can be found via inverses.

We start from the definition

$$*A057162 = *A069768 \circ *A057508 \quad (130)$$

hoping eventually to apply (31). As $*A057508 = (ENIPS *A057509)$ and $*A069768 = (ENIPS *A069770)$ and by (33) these translate as

$$(\text{define } (*A057508 \text{ s}) (\text{fold-right } (\text{lambda } (x \ y) \ (*A057509 \ (\text{cons } x \ y))) \ '() \ \text{s})) \quad (131)$$

and

$$(\text{define } (*A069768 \ \text{s}) (\text{fold-right } (\text{lambda } (x \ y) \ (*A069770 \ (\text{cons } x \ y))) \ '() \ \text{s})) \quad (132)$$

Then by plugging **A069768* as *h* and $(\text{lambda } (x \ y) \ (*A057509 \ (\text{cons } x \ y)))$ as *w* into (31) we get that **A057162* = *fold(m, (), s)* where *m* has been implicitly defined with a condition that

$$(\text{m } x \ (*A069768 \ y)) = (*A069768 \ (\text{lambda } (x \ y) \ (*A057509 \ (\text{cons } x \ y)))) \quad (133)$$

Then *m* should be massaged to the form $(\text{lambda } (x \ y) \ (g \ (\text{cons } x \ y)))$ and then *g* is our searched for bijection. Thus, applying **A069767* to the *y*-argument at both sides, and transferring **A069768* inside the lambda, we obtain:

$$(\text{define } (\text{m } x \ y) \ (*A069768 \ (*A057509 \ (\text{cons } x \ (*A069767 \ y))))) \quad (134)$$

Now, what happens above, is essentially that *x* is transferred to the leftmost tip of the binary tree *y*. This can be also done by surrounding **A057509* with a *more powerful* operation than **A069767/*A069768*, i.e. in this case, we can conjugate with **A057163* instead:

$$(\text{define } (\text{m } x \ y) \ (*A057163 \ (*A057509 \ (\text{cons } (*A057163 \ x) \ (*A057163 \ y))))) \quad (135)$$

(This works because **A057509* touches/traverses only along the top-level "spine" of the list, so it doesn't care what changes has happened elsewhere, and as we are conjugating, those extra-changes are eventually undone).

In other words, we are now performing **A057163*-conjugated *append*, as:

$$(\text{define } (\text{m } x \ y) \ (*A057163 \ (\text{append } (*A057163 \ y) \ (\text{list } (*A057163 \ x))))) \quad (136)$$

Thus, **A057162* can be defined as a following fold:

$$(\text{define } (*A057162 \ s) \ (\text{fold-right } \text{m } '() \ s)) \quad (137)$$

Bijection **A069773* (being **A057163*-conjugate of **A057501*) can be defined as:

$$\begin{aligned} &(\text{define } (*A069773 \ s) \\ & \quad (\text{cond } ((\text{null? } \ s) \ s) \\ & \quad \quad (\text{else } (*A057163 \ (\text{append } (*A057163 \ (\text{cdr } \ s)) \ (\text{list } (*A057163 \ (\text{car } \ s)))))) \\ & \quad) \\ &) \end{aligned} \quad (138)$$

which proves that

$$*A057162 = (ENIPS \ *A069773) \quad (139)$$

and thus by inverses, we also get:

$$*A057161 = (SPINE \ *A069774) \quad (140)$$

Bijection **A057503* is known as Emeric Deutsch's bijection "Gamma" on Dyck paths, and it can be defined as:

$$*A057503 = (ENIPS \ *A057501) \quad (141)$$

which definition of course can be opened, and represented as a fold:

$$(\text{define } (*A057503 \ s) \ (\text{fold-right } (\text{lambda } (x \ y) \ (\text{append } x \ (\text{list } y))) \ '() \ s)) \quad (142)$$

We observe that bijections $*A057162$ and $*A057503$ are $*A057164$ -conjugates of each other, that is, we have:

$$*A057162 = *A057164 \circ *A057503 \circ *A057164 \quad (143)$$

or equally:

$$*A057503 = *A057164 \circ *A057162 \circ *A057164 \quad (144)$$

multiplying the both sides of the latter from the left with $*A057164$, leaves us with:

$$*A057164 \circ *A057503 = *A057162 \circ *A057164 \quad (145)$$

and it is easily seen both sides produce an identical result tree from any tree they are given as an argument. (XXX – If one draws a few illustrations, that is!) It might help if we rewrite the right side as:

$$*A057164 \circ *A057503 = *A069768 \circ *A057508 \circ *A057164 \quad (146)$$

and because $*A057508 \circ *A057164 = *A057164 \circ *A057508 = (RIBS *A057164)$ we get:

$$*A057164 \circ *A057503 = *A069768 \circ (RIBS *A057164) \quad (147)$$

11.11. Automorphism $*A074679/*A074680$

Fixed points counted by: $LEFT(A019590)$.

Cycles counted by: $LEFT(A001683)$.

Max. cycle lengths given by: $A089410$.

LCM's of cycle lengths given by: $A089410$.

Lukasiewicz-word permuting: No.

Telescoping: No.

Compositions: $*A074679 = *A57163 \circ *A72796 \circ *A69770 \circ *A57163 \circ *A72796 = *A69770 \circ *A57163 \circ *A72796 \circ *A69770 \circ *A57163$

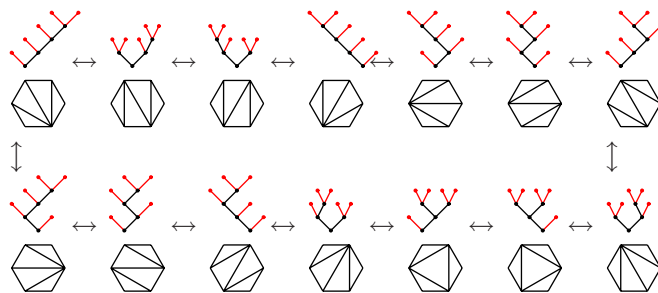
```
(define (*a074679! s)
  (cond ((pair? s) (cond ((pair? (cdr s)) (robl! s)) (else (swap! s))
                        s))
        s))
```

Destructive variants using primitives swap! and robl!/robr!:

```
(define (*a074680! s)
  (cond ((pair? s) (cond ((pair? (car s)) (robr! s)) (else (swap! s))
                        s))
        s))
```

Rotate binary tree, if possible, otherwise swap its sides.

How automorphism $*A074679/A074680$ will act transitively on the set of fourteen Catalan structures of size 4, forming a single cycle, is shown below.



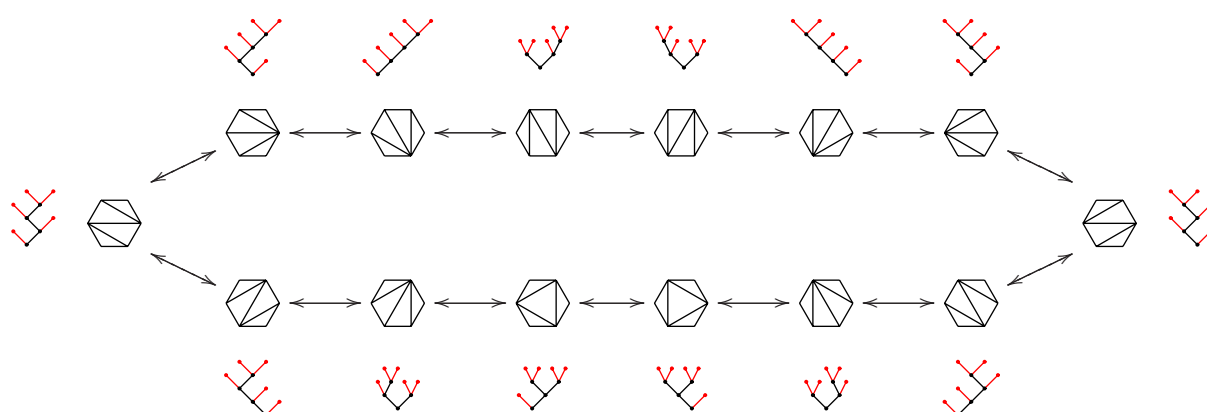


Figure 4: How automorphism $*A074679/A074680$ acts transitively on the set of fourteen binary trees of 4 internal nodes, and on the corresponding Eulerian polygon triangulations.

11.12. Automorphism $*A057501/*A057502$

Fixed points counted by: $LEFT(A019590)$.
Cycles counted by: $LEFT(A002995)$.
Max. cycle lengths given by: $A057543$.
.
Lukasiewicz-word permuting: No.
Telescoping: No.
Recursive compositions: $*A057501 = (SPINE *A74680)$
 $*A057502 = (ENIPS *A74679)$
Compositions: $*A057501 = *A57509 \circ *A69770$
 $*A057502 = *A69770 \circ *A57510 = *A69888 \circ *A82313$
Constructive variant for A057501 using Lisp/Scheme built-in function append:

```
(define (*a057501 s)
  (cond ((null? s) s)
        (else (append (car s) (list (cdr s))))))
(define (*a057501! s)
  (cond ((null? s))
        ((not (pair? (car s))) (swap! s))
        (else (robr! s) (*a057501! (cdr s))))
  s)
(define (*a057502! s)
  (cond ((null? s))
        ((not (pair? (cdr s))) (swap! s))
        (else (*a057502! (cdr s) (robl! s))))
  s)
(define (*a057501v2! s)
  (cond ((pair? s) (*a074680! s) (*a057501v2! (cdr s))))
  s)
(define (*a057502v2! s)
  (cond ((pair? s) (*a057502v2! (cdr s) (*a074679! s))))
  s)
```

Destructive variants using primitives swap! and robl!/robr!:

Destructive variants using automorphisms A074680 and A074679:

Rotate non-crossing chords (handshake) arrangements; Rotate root position of the general trees.

Proposition 12. The following two identities hold:

$$*A057501 = (SPINE *A074680) \tag{148}$$

$$*A057502 = (ENIPS *A074679) \tag{149}$$

Proof. We prove (149) by substituting $*A069770$ for f , and $*A072796$ for g in lemma 10. Then

$$\begin{aligned} h &= *A069770 \circ *A072796 \circ *A069770_{cdr}^{-1} \\ &= *A069770 \circ *A072796 \circ *A089850 \\ &= *A074679 \end{aligned} \tag{150}$$

The case (148) follows from the identity $(ENIPS f)^{-1} = (SPINE f^{-1})$.

Note. There is also a nice graphical proof for this.

The cycle counts are computed as

$$cc_{A057501}(n) = \frac{1}{2n} \sum_{d|n} \phi\left(\frac{n}{d}\right) \binom{2d}{d} - \frac{C(n)}{2} + (\text{if } n \text{ is odd}) \frac{C(\frac{n-1}{2})}{2} \quad (151)$$

yielding

$$1, 1, 1, 2, 3, 6, 14, 34, 95, 280, 854, 2694, 8714, 28640, 95640, 323396, \dots$$

which is the OEIS sequence A002995 shifted once left. Note that each cycle of automorphism **A057501* contains one or more cycles of automorphism **A057509*, that is, the latter automorphism *refines* the partitioning of the S-expressions effected by the former. Thus $cc_{A057501}(n) \leq cc_{A057509}(n)$ for all n , which can also be seen from the above formula, whose first term is actually $cc_{A057509}(n)$, i.e. A003239(n). Note that $A002995(n+1) = A003239(n) - A007595(n)$, when n is even and ≥ 2 . Of the set of $\{C_n$ structures of size $n\}$, all the $C(n-1)$ planted trees are fixed by automorphism **A057509*

That the identity $*A057501 = *A057509 \circ *A069770$ holds can be seen immediately by comparing those definitions of **A057501* and **A057509* that use the function `append`.

11.13. Automorphism **A057505/*A057506*

Fixed points counted by: *LEFT*(A019590).

Cycles counted by: A057507.

Max. cycle lengths given by: A057545.

LCM's of cycle lengths given by: A060114.

Lukasiewicz-word permuting: No.

Telescoping: No.

Recursive compositions: **A057505* = (KROF **A57501*)

**A057506* = (FORK **A57502*)

Compositions: **A057505* = **A57164* ◦ **A57163*

**A057506* = **A57163* ◦ **A57164*

```
(define (*a057505 a)
```

```
  (let loop ((a a) (b (list)))
```

```
    (cond ((not (pair? a)) b)
```

```
          (else (loop (car a) (cons (*a057505 (cdr a)) b))))))
```

Constructive, tail-recursive variants:

```
(define (*a057506 a)
```

```
  (let loop ((a a) (b (list)))
```

```
    (cond ((not (pair? a)) b)
```

```
          (else (loop (cdr a) (cons b (*a057506 (car a))))))
```

```
(define (*a057505v2 s)
```

```
  (cond ((null? s) s)
```

```
        (else (append (*a057505v2 (car s)) (list (*a057505v2 (cdr s))))))
```

Constructive, recursive variant for **A057505* using Lisp/Scheme built-in function append:

```
(define (*a057505v3 s)
```

```
  (with-input-from-string (list->string-strange s) read))
```

```
(define (list->string-strange s)
```

```
  (string-append
```

```
    "("
```

```
    (with-output-to-string
```

```
      (lambda ()
```

```
        (let recurse ((s s))
```

```
          (cond ((pair? s) (recurse (car s))
```

```
                  (write-string "(")
```

```
                  (recurse (cdr s))
```

```
                  (write-string "))))))
```

```
    ")"))
```

```
(define (*a057505! s)
```

```
  (cond ((pair? s) (*a057505! (car s)) (*a057505! (cdr s)) (*a057505! s))
```

Destructive variants, built on handshake rotates:

```
(define (*a057506! s)
```

```
  (cond ((pair? s) (*a057502! s) (*a057506! (car s)) (*a057506! s))
```

Donaghey's M.

$$A001405(n) \stackrel{\text{def}}{=} \binom{n}{\lfloor n/2 \rfloor} \quad (152)$$

$$A001683(n) \stackrel{\text{def}}{=} \frac{1}{n} C(n-2) + \frac{1}{2} c\left(\frac{n}{2} - 1\right) + \frac{2}{3} c\left(\frac{n}{3} - 1\right) \quad (153)$$

$$A002995(n) \stackrel{\text{def}}{=} \frac{1}{(2(n-1))} \sum_{d|(n-1)} \phi\left(\frac{n-1}{d}\right) \binom{2d}{d} - \frac{C(n-1)}{2} + (\text{if } n \text{ is even}) \frac{C(\frac{n}{2}-1)}{2} \quad (154)$$

$$A003239(n) \stackrel{\text{def}}{=} \frac{1}{(2n)} \sum_{d|n} \phi(n/d) \binom{2d}{d} \quad (155)$$

$$A007123(n) \stackrel{\text{def}}{=} \frac{C(n) + \binom{n}{\lfloor n/2 \rfloor}}{2} \quad (156)$$

$$A034731(n) \stackrel{\text{def}}{=} \sum_{d|n} C(d-1) \quad (157)$$

$$A073431(n) \stackrel{\text{def}}{=} \frac{1}{2^{n-1}} \sum_{i=1}^{2^{n-1}} \sum_{j=0}^{A007814(i)} A073346(n, j) \stackrel{\text{def}}{=} -1 + \frac{1}{2^{n-2}} \sum_{i=1}^{2^{n-1}} A073346(n, A007814(i)) \stackrel{\text{def}}{=} \frac{1}{2^n} \sum_{i=0}^n 2^{n-i} A073346(n, i) \stackrel{\text{def}}{=} \sum_{i=0..n} A074079(n, i) \quad (158)$$

12. Relations to other groups acting on other variants of binary trees

13. Open questions - XXX - vague ideas

- “Top-level” analysis of the automorphism group. For example, the whole group of automorphisms contains an alternating group (consisting only of automorphisms with even permutations, e.g. automorphism **A089851*, whose signature permutations contains only 3-cycles and fixed points.) Lukasiewicz-word permuting automorphisms form a subgroup of their own, and similarly initial-nil and other property reserving automorphisms. Automorphisms that leave top-level cdr-spine intact (ones formed with e.g. RIBS-transformation)? Can we say that the automorphisms that embed into Lukasiewicz-word permuting automorphisms in scale $n : 2n$ somehow form a subgroup? Yes, but of the whole group, not as a subgroup of L-word permuting automorphisms. The self-embedding of *binaryform trees* induces a group homomorphism from L-word preserving subgroup to whole automorphism group and still the homomorphism has a non-trivial kernel, because from any automorphism g one can construct a L-word preserving automorphism h where g embeds into, and this can be done in several possible (uncountably many) ways. Note that this homomorphism has at least two fixed points, the identity automorphism **A001477*, and automorphism **A072797*. XXX – Also **A069775* and **A069776*, to be proved.

It is easy to see that the subgroup of non-recursive automorphisms is not a normal subgroup. However, the latter itself contains further subgroups, e.g. all *form-preserving* non-recursive automorphisms of a single non-default clause with a particular tree shape of n leaves form a

subgroup isomorphic with symmetry group S_n of n elements. Thus there are $A000108(n - 1)$ subgroups isomorphic with S_n . (more?) Which of these subgroups are normal (with respect to non-recursive group)? Which compositions are closed? Delve into non-recursive automorphisms in more detail, in this same or separate paper.

- Automorphism group of Tamari-lattice (or actually, the corresponding *rotation graphs*) ? automorphism **A057163* is always one of the generators.
- Groups formed by using more than one Catalan automorphism as generators. E.g. various dihedral groups, composed of automorphism **A057161* and automorphism **A057163*, or automorphism **A057501* and automorphism **A057164*, etc. “Artificial dihedral” groups, like those composed of automorphism **A057511* and automorphism **A057164* or automorphism **A057509* and automorphism **A057508*?
- Actions on Stern-Brocot tree (previous section), etc.?
- Actions on *birds* (lambda-calculus)? C.f. *combinatorial ornithology*. “Simple programs”. Use Wolfram-esque plots.
- “Measures of chaoticity”? E.g. the sequences $A081164$ and $A081166$ for the automorphism **A057505*. First differences of the max-cycle sequences? (if not monotone) Sort the automorphisms according to the growth rate of their LCM-sequences. Involutions are then “by definition” the least chaotic? Compare the fixpoint/cycle count ratios for these, are there examples of diverging, “chaotic” behaviour? Again, chaoticity, can we define something like a “Lyapunov-exponent” for these? The underlying metric (whether a distance in Tamari-lattice, position in $A014486$, or in the cycle of some yet-to-be-found-natural-transitive-automorphism), will in any case favour some automorphisms over others. Develop a measure of chaoticity (a function) about which something predictable can be said when two automorphisms with known max-values or growth-rates of that function are composed. Then prove that some automorphisms are not like the others. No, not at all. Like Meeussen’s **A057117* and its bigger brother **A072088*, maybe. (Compute the b-files for many signature-permutations and their contractions and plot the graphs... E.g. compare the plot of $A082364$ with that of $A038776$. and $A082363$ with that of $A070041$ What do you see? Not much difference. Well, compute further...) At least $A072619$ ’s graph seems to have something like a seed for chaos...
- Analyze program time complexity ($O(\dots)$ and all that) measures? Depends on the “fundamental substratum” upon which the act is done. E.g. automorphism **A057164* is quadratic (*XXX - at least sometimes?*) when implemented as acting on S-expressions, while it is always linear when implemented directly to act on $A014486$ -codes.
- What is the most naturally defined automorphism that acts transitively on S-expressions of all sizes? I.e. whose cycle-count sequence is $A000012$ (all-one sequence)?
- Find previously unknown manifestations of Catalonia by searching through automatically generated automorphisms with modestly (e.g. linearly) growing LCM-sequences? (*or involutions.*) That is, find an automorphism of a structure before the structure itself!
- Nice visualizations of the global behaviour? E.g. using something like Tamari-lattice (or all binary trees arranged in a spiral fashion, etc.), and then drawing each cycle on it with a different colour, or at slightly different height, if a kind of 3D-visualization is used. Visualization which would highlight various instances self-embeddability?

Acknowledgements

The author thanks Wouter Meeussen for bringing the rich field of Catalan Automorphisms to his attention. He also thanks Mikromuseo ry. of Finland and TKSoft Oy Inc., the former for providing its DEC Alphastations for his computer runs, and the latter for letting him to polish this paper at their premises.

References

- [**Bernstein and Sloane 1995**] M. Bernstein and N. J. A. Sloane, Some canonical sequences of integers, *Linear Algebra and Its Applications*, vol. **226–228** (1995), 57–72.
- [**Callan 2004a**] D. Callan, Some bijections and identities for the Catalan and Fine numbers, *Sém. Lothar. Combin.* **53** (2004/06), Art. B53e, 16 pp.
- [**Callan 2004b**] D. Callan, Two bijections for Dyck path parameters, [math.CO/0406381](https://arxiv.org/abs/math.CO/0406381), 2004, 4pp.
- [**Callan 2006**] D. Callan, A Bijection on Dyck Paths and Its Cycle Structure, http://www.stat.wisc.edu/~callan/papers/Motzkin_manifestation/, 2006, 17pp.
- [**Cameron 1989**] P. J. Cameron, Some sequences of integers, *Discrete Math.*, **75** (1989), 89–102.
- [**Cameron 2000**] P. J. Cameron, Sequences realized by oligomorphic permutation groups, *J. Integ. Seqs.*, Vol. **3** (2000), #00.1.5.
- [**De Bruijn and Morselt 1967**] N.G. De Bruijn and B.J.M. Morselt, A note on plane trees, *J. Combin. Theory*, **2** (1967), 27–34.
- [**Deutsch 1999**] E. Deutsch, An involution on Dyck paths and its consequences, *Discrete Math.* **204** (1999), no. 1-3, 163–166.
- [**Deutsch 1998**] E. Deutsch, A bijection on Dyck paths and its consequences, *Discrete Math.* **179** (1998), no. 1-3, 253–256.
- [**Deutsch 2000**] E. Deutsch, A bijection on ordered trees and its consequences, *J. Combin. Theory Ser. A* **90** (2000), no. 1, 210–215.
- [**Deutsch and Elizalde**] Emeric Deutsch and Sergi Elizalde, A simple and unusual bijection for Dyck paths and its consequences, *Ann. Comb.* **7** (2003), no. 3, 281–297.
- [**Donaghey 1980**] R. Donaghey, Automorphisms on Catalan trees and bracketing, *J. Combin. Theory, Series B*, **29** (1980), 75–90.
- [**Er 1989**] M. C. Er, Classes of admissible permutations that are generatable by depth-first traversals of ordered trees, *The Computer Journal* **32** (1989), no. 1, 76–85.
- [**Gardner 197?**] Martin Gardner, XXX *Scientific American*
- [**Gibbons 2005**] Jeremy Gibbons, Metamorphisms: Streaming Representation-Changers, *Preprint submitted to Elsevier Science*

- [**Gibbons, Hutton and Altenkirch 2001**] Jeremy Gibbons, Graham Hutton and Thorsten Altenkirch, When is a function a fold or an unfold?, *Electronic Notes in Theoretical Computer Science* **44** (2001), no. 1, 14pp.
<http://www.elsevier.nl/locate/entcs/volume44.html>,
- [**Goebel 1980**] F. Goebel, On a 1-1-correspondence between rooted trees and natural numbers, *J. Combin. Theory, Series B*, **29** (1980), 141–143.
- [**Hutton 1999**] G. Hutton, A tutorial on the universality and expressiveness of fold, *J. Functional Programming*, **9** (July 1999), 355-372.
- [**Nikos K, 2001**] Nikos K., κρητικός αλριθμογαςτος, το ζωο φαντασμα,
<http://stigmis.gr/gr/grpages/articles/cat.htm>
- [**Knuth fascicle 4a**] D. Knuth, *Art of Computer Programming, Vol.4, Fascicle 4: Generating all Trees – History of Combinatorial Generation*, Addison-Wesley, 2006, vi+120pp, draft available from <http://www-cs-faculty.stanford.edu/~knuth/fasc4a.ps.gz>
- [**Kreweras 1970**] G. Kreweras, Sur les éventails de segments, *Cahiers du Bureau Universitaire de Recherche Opérationnelle, Cahier no. 15*, Paris, 1970, pp. 3-41.
- [**Lalanne 1992**] J.-C. Lalanne, Une involution sur les chemins de Dyck, *European J. Combin.* **13** (1992), no. 6, 477–487.
- [**Lalanne 1993**] J.-C. Lalanne, Sur une involution sur les chemins de Dyck, Conference on Formal Power Series and Algebraic Combinatorics *Theoret. Comput. Sci.* **117** (1993), no. 1-2, 203–215.
- [**Le Borgne 2006**] Y. Le Borgne, An algorithm to describe bijections involving Dyck paths, FPSAC 06, <http://garsia.math.yorku.ca/fpsac06/papers/46.pdf>, 2006, 12pp.
- [**Matula 1968**] D. Matula, A natural rooted tree enumeration by prime factorization, *SIAM Rev.*, **10** (1968).
- [**McCarthy 1960**] J. McCarthy, Recursive functions of symbolic expressions and their computation by machine, Part I, *Comm. A.C.M.*, **3** (1960), 184–195.
- [**Sloane 1995–**] N. J. A. Sloane, *The On-Line Encyclopedia of Integer Sequences*, published electronically at www.research.att.com/~njas/sequences/.
- [**Stanley 1999**] R. P. Stanley, *Enumerative Combinatorics*, Vol. 2, Cambridge University Press, 1999.
- [**Stanley 2001–**] R. P. Stanley, *Catalan addendum*, published electronically at <http://www-math.mit.edu/~rstan/ec/#catadd>
- [**Sussman and Steele 1975**] G. Sussman and G. Steele, SCHEME: An Interpreter for Extended Lambda Calculus, *AI Memo 349*, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, December 1975.
- [**Vaillé 1997**] J. Vaillé, Une bijection explicative de plusieurs propriétés remarquables des ponts, *European J. Combin.* **18** (1997), no. 1, 117–124.