

ON EFFICIENT BIJECTIONS BETWEEN PERMUTATIONS AND FUNCTIONS

LEEMON C. BAIRD III AND MICHAEL D. COLLINS

26 May 2006

U.S. Air Force Academy Technical Report

USAFA-TR-2007-02

ABSTRACT. We explore efficient surjections and bijections between sets of functions and sets of permutations. These mappings are efficient in the sense that with only a few evaluations (polynomial in the size of inputs) of a black box representing either a function or permutation, we can evaluate the corresponding permutation or function for a particular input. It is not obvious that such efficient surjections exist. We give several such surjections. Furthermore, we define a new subgroup of permutations, the mobile permutations, and show that our surjections are actually bijections between the set of mobiles and the set of all possible functions. Through the use of these bijections, we then derive several surprising results. In fact, all of these bijections and their inverses have very simple forms, all of which are efficient, and whose existence may be unexpected. For example, there is an efficient mapping from each mobile permutation to its inverse. Additionally, mobile permutations can be associated with trees which are, in turn, associated with the corresponding functions in a natural way. One of the most surprising facts is that these bijections between permutations over vectors, and these functions over vectors, are best understood by first considering functions over strings. This leads to the definition of left-dependent permutations and additional insights into this system. We conclude with ideas for further research suggested by these results.

1. INTRODUCTION

If G is a group, we consider permutations of \mathbf{G}^n , where \mathbf{G}^n is the direct product of \mathbf{G} with itself n times. In fact, we can enumerate all permutations of \mathbf{G}^n from 1 to $|\mathbf{G}|^n!$. We can also number all functions, $f : \mathbf{G}^n \rightarrow \mathbf{G}$, from 1 to $|\mathbf{G}|^{|\mathbf{G}|^n}$. Most simple surjections between these sets of indices will not be efficient. That is, given, π , a permutation of \mathbf{G} as a black box, it may take many evaluations of π to obtain enough information about its index to calculate $f(x)$. Is there an efficient surjection, ϕ , between permutations of \mathbf{G} , and this set of functions? In particular, is it possible to evaluate $\phi(\pi)(x)$ in just one evaluation of π ? The answer is yes!

We are looking for a mapping which is both surjective and efficient. For example, if $\mathbf{G} = \mathbb{Z}_2$ and \mathbb{Z}_2^n is the direct product of \mathbb{Z}_2 with itself n times, we can define a simple map from a given permutation to a Boolean function by letting $f(x)$ be the rightmost bit of $\pi(x)$, i.e., $f(x_n, x_{n-1}, \dots, x_1) = \pi(x_n, x_{n-1}, \dots, x_1)_1$. While this mapping is efficient, it is not a surjection. Simply note that because π is a permutation, $f(x) = \pi(x)_i$ is must be a balanced function. So this mapping cannot generate unbalanced Boolean functions.

Once we have such a surjection, a natural question is whether a simple subset of permutations can induce an efficient bijection for that mapping? Can the inverse mapping for such a bijection be efficient as well? Again, the answers are yes!

One particularly simple solution is given by $f(x) = \pi(x)_{R(x)}$, where $R(x)$ returns the position of the rightmost nonzero position in x . Notice that we calculate $f(x)$ by only one evaluation of π . By considering only permutations belonging to a subgroup called the Mobile Group, we derive a number of such bijections.

The construction of these bijections is unified by using functions over strings as a bridge between functions on tuples and the Mobile Group. In fact, functions on strings of group elements are group isomorphic to the Mobile Group. This isomorphism provides the means for efficient computation of inverses.

We quickly define our conventions and present a concrete example. We then look in-depth at the construction of our bijections. Using this construction, we illustrate a nice application. Finally, we discuss the automated computation of our bijections.

2. PRELIMINARIES: ORGANIZATION AND NOTATION

Figure 1 shows our bijections of interest. For $f \in \mathcal{F}$ we find $h \in \mathcal{H}$ depending on the edge we choose to follow. Then we compute the corresponding permutation, $\pi \in \mathcal{M}$ from h . Throughout this paper we will use the following conventions:

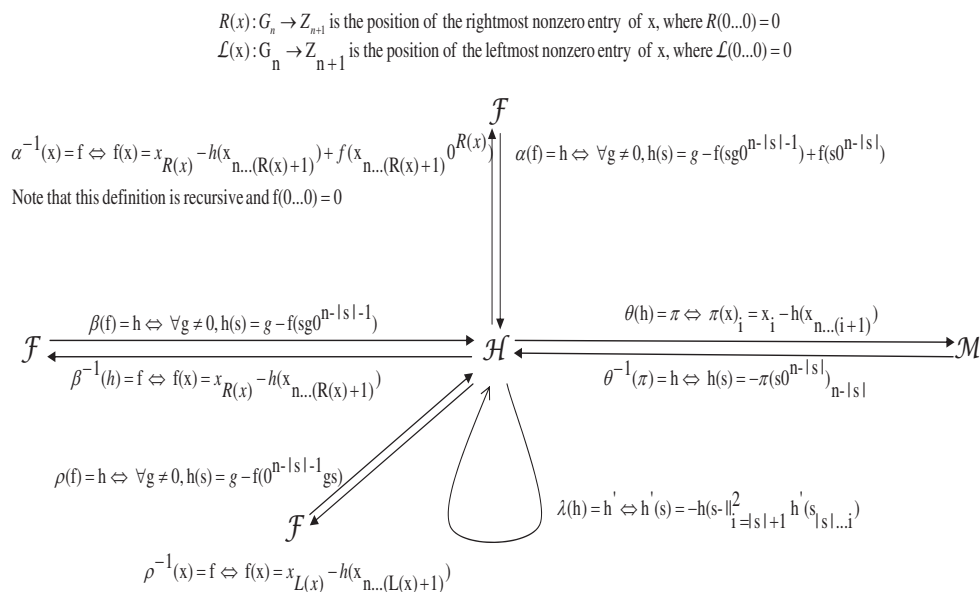
- (1) \mathbf{G} is a finite group with $|\mathbf{G}| = m$. \mathbf{G} is referred to as the base group.
- (2) \mathbf{G}^n is the direct product, $\mathbf{G} \times \mathbf{G} \cdots \times \mathbf{G}$, with $x = (x_n, \dots, x_1) \in \mathbf{G}^n$.
- (3) $\mathcal{F} = \left\{ f : \mathbf{G}^n \rightarrow \mathbf{G} \left| \begin{array}{l} f(0 \cdots 0) = 0, \\ \text{If } r = R(x) \text{ and if } \forall (g \in G) \ni (x_r + g \neq 0), \\ \text{then } f(x_{n \dots r+1}[x_r + g]0^{r-1}) = f(x_{n \dots r+1}[x_r]0^{r-1}) + g \end{array} \right. \right\}$
- (4) $\mathcal{H} = \left\{ h|h : \bigcup_{i=0}^{n-1} \mathbf{G}^i \rightarrow \mathbf{G} \right\}$, all functions from strings into \mathbf{G} .
- (5) $\Gamma = \{ \text{Strictly Left Dependent functions from } \mathbf{G}^n \rightarrow \mathbf{G}^n \}$. (see section 3).
- (6) $\mathcal{M} = \{ \pi | \pi(x) = x - \gamma(x) \text{ for some } \gamma \in \Gamma \}$.

While the condition on \mathcal{F} may seem stringent, if the base group is \mathbb{Z}_2 , then the only non-zero g is 1. Therefore, $g \oplus x_{R(x)} = 0$, which means the restriction is vacuously true. So, $\mathcal{F} = \{ f(x) | f : \mathbf{G}^n \rightarrow \mathbf{G}, f(0, \dots, 0) = 0 \}$ giving us a complete bijection between all possible Boolean functions (with $f(0, \dots, 0) = 0$) and the Mobile Group, $\mathcal{M}_{\mathbb{Z}_2}$.

A concrete example is instructive. Consider the middle path of bijections in Figure 1. We verify that $f(x) = \beta^{-1}(\theta^{-1}(\pi))(x) = \pi_{R(x)}$ for a specific f . Let \mathbb{Z}_2 be the base group with $n = 3$. Also let $f(x_3, x_2, x_1) = x_2x_1 \oplus x_3 \oplus x_2$. Hence,

$$\begin{aligned} \theta(\beta(f)) : \mathcal{F} &\rightarrow \mathcal{M} \\ \beta(f) = h : h(s) &= 1 \oplus f(s10^{n-|s|-1}) \\ \theta(h) = \pi : \pi(x)_i &= x_i \oplus h(x_{n \dots (i+1)}) \end{aligned}$$

The first two columns of Figure 2 are just the truth table for f . Columns 3 and 4 show the computation of h corresponding to $\beta(f)$. Column 5 shows the



A sequence of elements of G , x , is viewed both as an element of G^n and as a string. The notation x_i means the i^{th} element in the sequence, where the rightmost element is in position 1 and the leftmost element is in position n . The notation $x_{i\dots j}$ means the elements in the sequence from i to j with x_i being the least significant position. If $j > i$, $x_{i\dots j}$ is the empty string. When referring to variables which represent strings, adjacency denotes concatenation of strings. A superscript on a string will be an integer which denotes the number of repetitions of the string. For example, if x is a string and $x = abcd$, then $x0^4$ means $abcd0000$ and $(x0)^2$ means $abcd0abcd0$.

FIGURE 1. Map of Bijections

computation of π derived from $\theta(h)$. The last column shows that the composition of β^{-1} and θ^{-1} is $\pi(x)_{R(x)} = f(x)$.

We can graphically represent θ using trees as shown below. Each node in the tree is indexed in a canonical manner to the strings of length less than n . The value of the node is $h(s)$.

$$h(\emptyset) = 0$$

$$h(0) = 0$$

$$h(1) = 1$$

$$h(00) = 1 \quad h(01) = 1 \quad h(10) = 0 \quad h(11) = 0$$

$$\begin{array}{cccccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ 001 & 000 & 011 & 010 & 110 & 111 & 100 & 101 \end{array}$$

x	f	s	$h(s) = 1 \oplus f(s10^{n- s -1})$	$\pi(x)_i = x_i \oplus h(x_{n\dots(i+1)})$	$\pi(x)_{R(x)}$
000	0	—	—	$(0, 0, 1) \begin{cases} 0 \oplus h(\emptyset) = 0 \\ 0 \oplus h(0) = 0 \\ 0 \oplus h(00) = 1 \end{cases}$	0
001	0	\emptyset	$1 \oplus f(100) = 0$	$(0, 0, 0) \begin{cases} 0 \oplus h(\emptyset) = 0 \\ 0 \oplus h(0) = 0 \\ 1 \oplus h(00) = 0 \end{cases}$	0
010	1	0	$1 \oplus f(010) = 0$	$(0, 1, 1) \begin{cases} 0 \oplus h(\emptyset) = 0 \\ 1 \oplus h(0) = 1 \\ 0 \oplus h(01) = 1 \end{cases}$	1
011	0	1	$1 \oplus f(110) = 1$	$(0, 1, 0) \begin{cases} 0 \oplus h(\emptyset) = 0 \\ 1 \oplus h(0) = 1 \\ 1 \oplus h(01) = 0 \end{cases}$	0
100	1	00	$1 \oplus f(001) = 1$	$(1, 1, 0) \begin{cases} 1 \oplus h(\emptyset) = 1 \\ 0 \oplus h(1) = 1 \\ 0 \oplus h(10) = 0 \end{cases}$	1
101	1	01	$1 \oplus f(011) = 1$	$(1, 1, 1) \begin{cases} 1 \oplus h(\emptyset) = 1 \\ 0 \oplus h(1) = 1 \\ 1 \oplus h(10) = 1 \end{cases}$	1
110	0	10	$1 \oplus f(101) = 0$	$(1, 0, 0) \begin{cases} 1 \oplus h(\emptyset) = 1 \\ 1 \oplus h(1) = 0 \\ 0 \oplus h(11) = 0 \end{cases}$	0
111	1	11	$1 \oplus f(111) = 0$	$(1, 0, 1) \begin{cases} 1 \oplus h(\emptyset) = 1 \\ 1 \oplus h(1) = 0 \\ 1 \oplus h(11) = 1 \end{cases}$	1

FIGURE 2. Computing $\pi(x)$ for $x_2x_1 \oplus x_3 \oplus x_2$

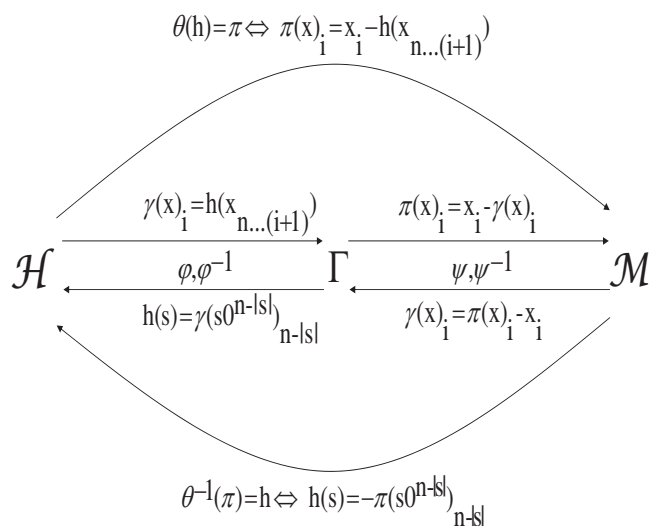
The two bottom rows explicitly define the resulting permutation. The leaves are initially labeled by the elements in the domain of f . The second row shows the mapping induced by the values in the tree nodes. We obtain the second row from the first by addition in \mathbf{G}^n . The element in \mathbf{G}^n induced by the values of the nodes in the path above a leaf serves as the additive for that leaf. For example, the path above 011 induces the additive 001, so that 011 is mapped to 010 by the computation $011 \oplus 001 = 010$. Similarly, the path above the leaf labeled 101 induces the additive 010 so that 101 is mapped to 111.

The permutation corresponding to the function $g(x) = 0$, is the identity permutation and the two rows would be identical.

The bijections in Figure 1 are unified by θ . In fact, θ induces a group isomorphism between \mathcal{M} and \mathcal{H} . This isomorphism allows us to compute inverses in \mathcal{M} very efficiently. The bijection θ is derived from the concept of Strictly Left Dependent functions. We now introduce these functions, prove several interesting properties, and construct the induced isomorphism.

3. THE FUNDAMENTAL BIJECTION

Figure 3 serves as a road map for this discussion. In particular, we define ψ and prove it is a bijection. We then define the Mobile Group and show θ is a bijection.

FIGURE 3. The Fundamental Bijection: θ

3.1. Strictly Left Dependent Functions. A function, $\gamma : \mathbf{G}^n \rightarrow \mathbf{G}^n$, is **Strictly Left Dependent**, SLD, iff each output position of γ is a function of only the input positions to the left of that position. More specifically, we define a Strictly Left Dependent function as follows.

$$\gamma \text{ is SLD} \iff \forall 1 \leq i \leq n, w \in \mathbf{G}^i \text{ and } x \in \mathbf{G}^n, [\gamma(x)_i = \gamma(x_{n...(i+1)}w)_i] \quad (3.1)$$

We define Left Dependent functions, LD, similarly.

$$\gamma \text{ is LD} \iff \forall 1 \leq i \leq n, w \in \mathbf{G}^{(i-1)} \text{ and } x \in \mathbf{G}^n, [\gamma(x)_i = \gamma(x_{n...(i)}w)_i] \quad (3.2)$$

A few immediate properties about SLD functions are apparent.

Remark 3.1. The sum of SLDs is SLD. If the output at each position depends only on the positions to the left, then the component-wise addition of two such functions cannot change that property.

Remark 3.2. The composition of an SLD function and an LD function is SLD. If $\gamma : \mathbf{G}^n \rightarrow \mathbf{G}^n$ is SLD and $\pi : \mathbf{G}^n \rightarrow \mathbf{G}^n$ is LD, then $\gamma(\pi(x))_i$ depends only on the elements $\pi(x)_n, \dots, \pi(x)_{i+1}$. Additionally, π LD implies that each element $\pi(x)_j$, $(i+1) \leq j \leq n$, depends on at most the positions j through n . So, $\gamma(\pi(x))_i$ depends only on x_n, \dots, x_{i+1} and so $\gamma \circ \pi$ is SLD. Analogous arguments show that $\pi \circ \gamma$ is SLD, and that the composition of LD functions is LD.

We can also count the number of Strictly Left Dependent functions for a given \mathbf{G}^n . Let $\Gamma_{\mathbf{G}^n} = \{\gamma : \mathbf{G}^n \rightarrow \mathbf{G}^n \text{ where } \gamma \text{ is SLD}\}$. Since every output position of an SLD depends only on the input positions to the left of that position, we see that there are $|\mathbf{G}|^{|\mathbf{G}|^{n-i}}$ definitions for $\gamma(x)_i$. So if $|\mathbf{G}| = m$,

$$|\Gamma_{\mathbf{G}^n}| = m^{\sum_{i=0}^{n-1} m^i} = |\mathbf{G}|^{\frac{m^n - 1}{m - 1}} \quad (3.3)$$

Recalling from Figure 1 that $\mathcal{L}(x) : \mathbf{G}^n \rightarrow \mathbb{Z}_{n+1}$ is the left-most non-zero position of $x \in \mathbf{G}^n$, where $\mathcal{L}(0) = 0$, we can alternatively characterize SLD functions as follows.

Theorem 3.3.

$$\gamma : \mathbf{G}^n \rightarrow \mathbf{G}^n \text{ is SLD} \iff \forall x, y \in \mathbf{G}^n, x \neq y, \mathcal{L}(\gamma(x) - \gamma(y)) < \mathcal{L}(x - y)$$

Proof: \Rightarrow . Suppose that $x \neq y$ and that $\mathcal{L}(x - y) = i$, then we want to show that $\mathcal{L}(\gamma(x) - \gamma(y)) < i = \mathcal{L}(x - y)$. By definition of \mathcal{L} , $\mathcal{L}(x - y) = i \Rightarrow x_{n \dots (i+1)} = y_{n \dots (i+1)}$. Since γ is SLD, this implies that $\gamma(x)_{n \dots i} = \gamma(y)_{n \dots i}$. Hence, $\gamma(x)$ and $\gamma(y)$ agree in all positions left of position $(i-1)$ and so $\mathcal{L}(\gamma(x) - \gamma(y)) < i = \mathcal{L}(x - y)$.

\Leftarrow . Now suppose that for γ , $x \neq y \Rightarrow \mathcal{L}(\gamma(x) - \gamma(y)) < \mathcal{L}(x - y) = i$. We show that γ is SLD. By definition of \mathcal{L} , $\mathcal{L}(x - y) = i$ implies that x and y match in each position left of position i . That is, $x_{n \dots (i+1)} = y_{n \dots (i+1)}$. So, by assumption, we have $\mathcal{L}(\gamma(x) - \gamma(y)) \leq (i-1)$. This means that $\gamma(x)$ and $\gamma(y)$ match in each position to the left of $(i-1)$. So $\gamma(x)_{n \dots i} = \gamma(y)_{n \dots i}$. Thus, w.l.o.g., we let $y = x_{n \dots (i+1)} w$ for some $w \in \mathbf{G}^i$ which means that $\gamma(x)_i = \gamma(x_{n \dots (i+1)} w)_i$. □

3.2. The Mobile Group of a finite group. We associate each Strictly Left Dependent function with a specific permutation on \mathbf{G}^n via the mapping ψ . The range of ψ is a group under composition and called the Mobile Group, $\mathcal{M}_{\mathbf{G}^n}$.

Theorem 3.4. $\gamma : \mathbf{G}^n \rightarrow \mathbf{G}^n$ is SLD $\Rightarrow \pi(x) = x - \gamma(x)$ is a permutation on \mathbf{G}^n

Proof: Since the domain and range of π are equal, it is sufficient to show that π is 1-1.

$$\begin{aligned} x, y \in \mathbf{G}^n, x \neq y &\Rightarrow [\mathcal{L}(x - y) \neq \mathcal{L}(\gamma(x) - \gamma(y))] \\ &\Rightarrow [(x - y) \neq \gamma(x) - \gamma(y)] \\ &\Rightarrow [x - \gamma(x) \neq y - \gamma(y)] \\ &\Rightarrow [\pi(x) \neq \pi(y)] \end{aligned}$$

□

Theorem 3.5. The set $\mathcal{M}_{\mathbf{G}^n} = \{\pi(x) = x - \gamma(x) \mid \gamma \text{ is SLD}\}$ is a group under composition.

Closure: If π_1 and π_2 are in $\mathcal{M}_{\mathbf{G}^n}$ with $\pi_1 = x - \gamma_1(x)$ and $\pi_2(x) = x - \gamma_2(x)$, then $\pi_2(\pi_1(x)) = x - \gamma_1(x) - \gamma_2(\pi_1(x))$. Therefore, remarks 3.1 and 3.2 imply $\pi_2 \circ \pi_1$ is SLD. So, composition of Mobile permutations is closed.

The identity permutation is in $\mathcal{M}_{\mathbf{G}^n}$. If $\gamma(x) = 0, \forall x \in \mathbf{G}^n$, then $\pi(x) = x$.

Associativity is induced from the fact that the composition is closed and composition of permutations is associative.

For every $\pi \in \mathcal{M}_{\mathbf{G}^n}$, there is a $\pi^{-1} \in \mathcal{M}_{\mathbf{G}^n}$ with $\pi \circ \pi^{-1}$ equal to the identity permutation. Since $\forall t, \pi^t(x) \in \mathcal{M}_{\mathbf{G}^n}$, there is a k such that $\pi^k(x)$ is the identity permutation. This means that $\pi^{k-1}(x) \in \mathcal{M}_{\mathbf{G}^n}$ is π^{-1} . □

3.3. θ is a bijection. Using Figure 3 as a guide, we now show θ is a bijection, by verifying that $\theta = \psi(\phi)$ and that both ϕ and ψ are bijections.

Definition 3.6. Let $\phi : \mathcal{H}_{\mathbf{G}^n} \rightarrow \Gamma_{\mathbf{G}^n}$. $\phi(h) = \gamma$ iff $\forall x \in \mathbf{G}^n, \gamma(x)_i = h(x_{n \dots (i+1)})$.

Lemma 3.7. ϕ is a bijection.

Proof: We prove that the range and domain have the same size and that ϕ is 1-1.

Equation 3.3 and the definition of $\mathcal{H}_{\mathbf{G}^n}$ imply that if $m = |\mathbf{G}|$ then

$$|\Gamma_{\mathbf{G}^n}| = |\mathbf{G}|^{\frac{m^n - 1}{m - 1}} = m^{\sum_{i=0}^{n-1} m^i} = |\mathcal{H}_{\mathbf{G}^n}| \quad (3.4)$$

Now assume that $h_1, h_2 \in \mathcal{H}_{\mathbf{G}^n}$, where $\gamma_A = \phi(h_1)$, $\gamma_B = \phi(h_2)$, and $h_1 \neq h_2$. Then, there exists a string $x_{n \dots (k+1)} \in \bigcup_{i=0}^{n-1} \mathbf{G}^i$ for which $\gamma_A(x)_k = h_1(x_{n \dots (k+1)}) \neq h_2(x_{n \dots (k+1)}) = \gamma_B(x)_k$. Therefore, $\gamma_A \neq \gamma_B$. □

Definition 3.8. $\psi : \Gamma_{\mathbf{G}^n} \Rightarrow \mathcal{M}_{\mathbf{G}^n}$, where $\forall i, \psi(\gamma)(x)_i = x_i - \gamma(x)_i$

Theorem 3.9. ψ is a bijection.

It is sufficient to show ψ is 1-1 because $\mathcal{M}_{\mathbf{G}^n}$ is defined as the range of ψ .

Suppose that $\psi(\gamma_1) = \psi(\gamma_2)$, with $\gamma_1 \neq \gamma_2$. Then,

$$\begin{aligned} \forall i, x_i - \gamma_1(x)_i = x_i - \gamma_2(x)_i &\Rightarrow [0 = \gamma_1(x)_i - \gamma_2(x)_i] \\ &\Rightarrow [\gamma_1 = \gamma_2] \Rightarrow \Leftarrow \end{aligned}$$

So ψ is 1-1. □

Corollary 3.10. $\theta = \psi(\phi)$ is a bijection.

For completeness, we verify the inverse bijection ϕ^{-1} by computing the composition $\phi^{-1}(\phi(h))(s)$ where $\phi(h) = \gamma$.

$$\phi^{-1}(\phi(h))(s) = h\left(\left[s0^{n-|s|}\right]_{n \dots n-|s|+1}\right) \text{ by definition of } \phi \quad (3.5)$$

The argument of h simplifies to s , so $\phi^{-1}(\phi(h))(s) = h(s)$.

Finally, we can also manually verify θ^{-1} as follows: Given $\theta(h)(x) = \pi(x)$, with $\pi(x)_i = x_i - h(x_{n \dots (i+1)})$, we compute $h(s) = (\theta^{-1} \circ \pi)(s)$ by letting $i = n - |s|$ and $x = s0^i$. So

$$\begin{aligned} h(s) &= -((s0^i)_i - h((s0^i)_{n \dots (i+1)})) \\ &= -(0 - h(s)) \\ &= h(s) \end{aligned}$$

We now use θ for a very nice application of our bijections, allowing us to introduce and study another mapping from Figure 1, λ .

4. AN APPLICATION OF BIJECTIONS:
COMPUTING $\pi^{-1}(x)$ WITH PARTIAL INFORMATION.

Suppose a permutation, π , from $\mathcal{M}_{\mathbb{Z}_2^3}$ has been chosen, and further suppose that you are asked to calculate $\pi^{-1}(101)$. You are allowed to know $\pi(x)$ for any x you choose. What is the fewest number of inputs which will guarantee your ability to calculate $\pi^{-1}(101)$? What specific values of inputs should you choose? It turns out the answer is 3 in this case and we can also find the specific input values required.

We answer this question using our bijections. In particular, θ induces an isomorphism between \mathcal{M} and \mathcal{H} . This, in turn, induces a binary operation, $*$, on members of \mathcal{H} . The mapping which takes $h \in \mathcal{H}$ to h^{-1} in which "inverse" is defined with respect to $*$ turns out to be very efficient. By employing Mathematica¹ to compute just the right composition of bijections we find the answer to our question.

We obtain an isomorphism by imposing the necessary requirements. The induced binary operator on strings corresponds to composition of Mobile permutations.

Definition 4.1. $(h_1 * h_2)(s) \triangleq \theta^{-1}(\pi_1 \circ \pi_2)(s) = \theta^{-1}(\theta(h_1) \circ \theta(h_2))(s)$ where $*$: $\mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$, $\pi_1 = \theta(h_1)$, and $\pi_2 = \theta(h_2)$. (\circ means composition.)

Using this definition, we manually compute this new operator. Notice that it is efficient as well.

Recall that $\|$ means concatenation of strings. Further, we are using the $s_{j\dots i}$ notation (see Figure 1) and concatenating from left to right while indexing string characters from right to left, so the top index is smaller than the bottom index. Recall from Figure 1 that if $i > j$, then $s_{j\dots i}$ is \emptyset . Additionally, the difference of two strings is the componentwise difference of the individual group elements in the two strings.

Theorem 4.2. $(h_1 * h_2)(s) = h_2(s) + h_1(s - \|\|_{j=|s|+1}^2 [h_2(s_{|s|\dots(j)})])$

Proof: By definition, if $\pi_1(x)_i = x_i - h_1(x_{n\dots(i+1)})$ and $\pi_2(x)_j = x_j - h_2(x_{n\dots(j+1)})$, then

$$\begin{aligned} \pi_1(\pi_2(x))_i &= \pi_2(x)_i - h_1(\pi_2(x)_{n\dots(i+1)}) \\ &= x_i - h_2(x_{n\dots(i+1)}) - h_1(\|\|_{j=n}^{i+1} \pi_2(x)_j) \\ &= x_i - h_2(x_{n\dots(i+1)}) - h_1(\|\|_{j=n}^{i+1} [x_j - h_2(x_{n\dots(j+1)})]) \end{aligned}$$

To compute $h_1 * h_2$ we need to apply θ^{-1} to the above permutation. We do that as follows:

First, recall that for any Mobile permutation π , $\theta^{-1}(\pi)(s) = -\pi(s0^{n-|s|})_{n-|s|}$. This implies that, for $\pi = (\pi_1 \circ \pi_2)$, we can let $x = s0^{n-|s|}$ and $i = n - |s|$, hence:

$$\begin{aligned} \theta^{-1}(\pi)(s) &= -((s0^i)_i - h_2((s0^i)_{n\dots(i+1)}) - h_1(\|\|_{j=n}^{n-|s|+1} [(s0^i)_j - h_2((s0^i)_{n\dots(j+1)})])) \\ &= -(h_2(s) - h_1(\|\|_{j=n}^{n-|s|+1} [(s0^i)_j - h_2((s0^i)_{n\dots(j+1)})])) \end{aligned}$$

¹Mathematica Vers. 5.1.1.0, Wolfram Research, Inc. copyright 1988-2005.

Noting that position j of x is precisely position $j - (n - |s|)$ of s , we can simplify the above to:

$$\begin{aligned}\theta^{-1}(\pi)(s) &= h_2(s) + h_1(\|_{j=|s|}^1[s_j - h_2(s_{|s|\dots(j+1)})]) \\ &= h_2(s) + h_1((s_{|s|} - h_2(\emptyset))(s_{|s|-1} - h_2(s_{|s|})) \cdots (s_1 - h_2(s_{|s|\dots 2}))) \\ &= h_2(s) + h_1(s - \|_{j=|s|+1}^2[h_2(s_{|s|\dots j})])\end{aligned}$$

□

$(\mathcal{H}, *)$ is a group under $*$, since we have defined $*$ to be the required operator for θ to be a group isomorphism. It is helpful, however, to observe a few basic features of this group without transiting back to $\mathcal{M}_{\mathbf{G}^n}$.

The identity function for the group $(\mathcal{H}, *)$ is the function $I \in \mathcal{H}$ where $\forall s \in \bigcup_{i=0}^{n-1} \mathbf{G}^i$, $I(s) = 0$. This function corresponds to the identity permutation via θ .

To see that each group element has an inverse, we compute the inverses using the fact that if $\forall s$, $(h_1 * h_2)(s) = 0$ then $h_2 = h_1^{-1}$ under $*$. We recursively define

$$h'(s) = -h(s - \|_{i=|s|+1}^2 h'(s_{|s|\dots i})), \text{ where } h'(\emptyset) = -h(\emptyset) \quad (4.1)$$

then $\forall s$,

$$\begin{aligned}(h * h')(s) &= h'(s) + h(s - \|_{i=|s|+1}^2 [h'(s_{|s|\dots i})]) \\ &= -h(s - \|_{i=|s|+1}^2 h'(s_{|s|\dots i})) + h(s - \|_{i=|s|+1}^2 [h'(s_{|s|\dots i})]) \\ &= 0\end{aligned}$$

Hence, $h' = h^{-1}$.

This means we can now define the inverse map, λ , with regard to \mathcal{H} , formally.

Definition 4.3. $\lambda : \mathcal{H} \rightarrow \mathcal{H}$, where $\lambda(h)(s) = -h(s - \|_{i=|s|+1}^2 (\lambda(h))(s_{|s|\dots i}))$ with $\lambda(h)(\emptyset) = -h(\emptyset)$.

Using λ we can compute π^{-1} for $\pi \in \mathcal{M}$ as $\pi^{-1} = \theta(\lambda(\theta^{-1}(\pi)))$. The recursive definition of λ makes manual computation cumbersome. The result of this composition via Mathematica is:

$$\pi^{-1}(x)_i = -\pi(\pi^{-1}(x)_{n \dots (i+1)}[-x_i]0^{i-1})_i \quad (4.2)$$

Equation 4.2 lets us answer the question we posed at the start of this section. For $\mathcal{M}_{\mathbb{Z}_2^3}$, we only need to know at most 3 values of π to compute $\pi^{-1}(101)$. The specific example is shown below. In the tree below, we show the the mobile permutation we have chosen. We then use equation 4.2 to compute $\pi^{-1}(101)$.

$$h(\emptyset) = 1$$

$$h(0) = 1$$

$$h(1) = 0$$

$$h(00) = 0 \quad h(01) = 1 \quad h(10) = 1 \quad h(11) = 0$$

$$\begin{array}{cccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ 110 & 111 & 101 & 100 & 001 & 000 & 010 & 011 \end{array}$$

We let $x = 101$ and so,

$$\pi^{-1}(101)_3 = \pi(100)_3 = 0$$

$$\pi^{-1}(101)_2 = \pi(000)_2 = 1$$

$$\pi^{-1}(101)_1 = \pi(011)_1 = 0$$

Which means $\pi^{-1}(101) = (010)$ as is shown in the tree. So, we only needed to know x and three images of π to compute $\pi^{-1}(x)$.

5. COMPUTING BIJECTIONS

We return to our original questions in regard to the efficiency of our mappings. In fact, we did compute the compositions of a large variety of bijections found in Figure 1. Most of those computations employed Mathematica. Some were verified by hand. Almost all result in very efficient mappings! See Appendix A for summary of the results. Appendix B contains the Mathematica source code.

One of the most surprisingly simple bijections results from computing $\beta^{-1}(\theta^{-1}(\pi))(x)$ from \mathcal{M} to \mathcal{F} (see Figure 1).

Theorem 5.1. $f(x) = \beta^{-1}(\theta^{-1}(\pi))(x) = \pi(x)_{R(x)}$.

Proof: Since θ is a bijection, we know that there exists an h such that $\forall i, \pi(x)_i = x_i + h(x_{n \dots (i+1)})$. Therefore, $h(x_{n \dots (R(x)+1)}) = x_{R(x)} - \pi(x_{n \dots R(x)} 0^{R(x)-1})_{R(x)}$. Furthermore, $\beta^{-1}(h)(x) = x_{R(x)} - h(x_{n \dots (R(x)+1)})$ so,

$$\begin{aligned} f(x) &= x_{R(x)} - h(x_{n \dots (R(x)+1)}) \\ &= x_{R(x)} - x_{R(x)} + \pi(x_{n \dots R(x)} 0^{R(x)-1})_{R(x)} \\ &= \pi(x)_{R(x)} \end{aligned}$$

□

Our code defines each of the bijections $\psi, \psi^{-1}, \phi, \phi^{-1}, \alpha, \alpha^{-1}, \beta, \beta^{-1}, \rho$, and ρ^{-1} . The code then verifies that each mapping is a bijection and that the inverse mappings are correct.

In the case of checking inverses for α , we found it easier to complete the verification by hand from the intermediate results. Specifically, in checking $f \xrightarrow{\alpha} h \xrightarrow{\alpha^{-1}} f$, if $f'(x) = \alpha^{-1}(\alpha)(x)$, then our code reduces the problem to further simplifying:

$$f'(x) = f(x) - f(R_0(x)) + f'(R_0(x)) \quad (5.1)$$

where $R_0(x)$ is the mapping which sets the element $x_{R(x)}$ to 0.

In order to prove that equation 5.1 implies $f'(x) = f(x)$, an induction argument suffices.

Theorem 5.2. $f'(x) = f(x) - f(R_0(x)) + f'(R_0(x)) \Rightarrow f'(x) = f(x)$

Proof: We induct on $R(x)$. $R(x) = n$ implies that only position n in x is non-zero.

So,

$$\begin{aligned} f'(x) &= f(x) - f(R_0(x)) + f'(R_0(x)) \\ &= f(x) - f(0) + f'(0) \\ &= f(x) \end{aligned} \quad (\text{Recall that } f(0) = 0 \forall f \in \mathcal{F})$$

Assume that $R(x) > k \Rightarrow f'(x) = f(x)$. Then, given x with $R(x) = k$, we have,
 $f'(x) = f(x) - f(R_0(x)) + f'(R_0(x))$ with $R(R_0(x)) > R(x)$, by definition of $R_0(x)$
 $= f(x)$ $f'(R_0(x)) = f(R_0(x))$ by assumption.

□

To check the inverse in the other direction simply note that the above theorem and the fact that both mappings are 1-1 provide sufficient proof.

6. GENERALIZED FEISTELS

One final surprise is worthy of note. There is a relationship between the concepts we have presented so far and Feistel ciphers. In fact, many of the basic concepts would arise naturally if one were to ask "what would a generalization of a Feistel cipher look like?" It turns out that the set of maximally-general Feistel rounds is equivalent to the set of the Mobile permutations with transpositions.

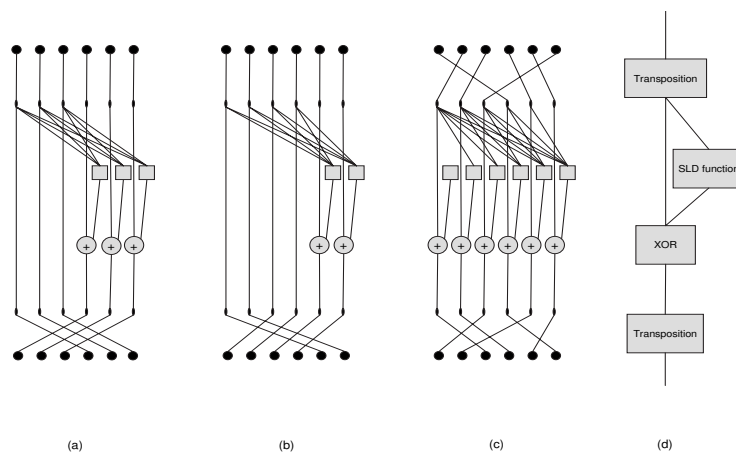


FIGURE 4. (a) a balanced Feistel, (b) an unbalanced Feistel, (c) a generalized Feistel, and (d) a block diagram of a generalized Feistel

Figure 4(a) shows a traditional, balanced Feistel round. The bits come in from the top, the left half is simply copied, the right half is XOR'd with some arbitrary function of the left half, and finally the halves are swapped. The result is a permutation on the input bit vector. If the functions are chosen according to a round key, then the composition of many such permutations form a Feistel cipher, of which DES is the most famous example.

Figure 4(b) shows a generalization of the Feistel. In an unbalanced Feistel, some fraction of the bits are XOR'd with a function of the remaining bits, and the result is shifted so that multiple rounds will eventually modify all the bits. In the example, only 2 bits change, dependent on the values of the other 4 bits.

How could a Feistel cipher be generalized as much as possible? In a Feistel round, at least one bit must pass through either unchanged, or XORed with a constant. Otherwise there would be no way to invert the round, since none of the arbitrary functions are guaranteed to be invertible. Once that bit has been chosen, there must be some other bit that is XORed with a bit that is solely a function the first bit. Then there must be a bit that is XORed with the result of a function of the first two bits, and so on. This process will give the most general form of Feistel possible, of which balanced and unbalanced Feistels are special cases.

Figure 4(c) shows this generalized Feistel round. Note that the first (leftmost) bit is XOR'd with a constant. The second bit is XOR'd with a function of the first bit. The n th bit is XOR'd with a function of the first $n - 1$ bits. This operation is preceded and followed by arbitrary transpositions.

Each of the functions in Figure 4(c) maps several input bits to a single output bit. The entire row of functions could be considered as a single function from \mathbb{Z}_2^n to \mathbb{Z}_2^n , where each output is dependent only on inputs to the left. Thus, the function can be any Strictly Left Dependent function. The output of the SLD function is then XOR'd with the input. That combination yields exactly the set of Mobile permutations, as shown in the previous sections. See Figure 4(d). Thus, the set of Mobile permutations, when preceded and followed by arbitrary transpositions, is equal to the set of permutations achievable by a generalized Feistel round.

7. CONCLUSIONS

We have constructed the Mobile Group of permutations which is a subgroup of permutations allowing bijections from \mathcal{F} to itself. These bijections are efficient in both directions. Of particular note is that the more canonical setting for these bijections is not \mathcal{F} but \mathcal{H} , functions on strings.

We have also used our new knowledge to evaluate inverse values of Mobile permutations efficiently which is, in general, not possible in the Symmetric group. We have also computed a wide variety of bijections arising from this analysis which should prove quite useful in the future.

Finally, we have shown the equivalence of the set of Mobile permutations with the set of permutations from a generalized Feistel round.

All of this encourages future exploration of these permutations.

8. FUTURE WORK

We plan to continue our analysis by trying to characterize various families of functions as permutations in the Mobile Group.

We are also quite interested in the pre-image problem. That is, can we characterize the set of Symmetric Group pre-images of a specific $f \in \mathcal{F}$ with a simple representation?

We are also interested in seeking out more surprising applications of these mappings.

APPENDIX A: Some Computed Bijections

Note: where applicable, $g \in \mathbf{G}$ and $g \neq 0$.

$$\begin{array}{ll}
\text{Compose: } f \xrightarrow{\alpha} h \xrightarrow{\theta} \pi & \pi(x)_i = x_i - g + f(x_{n\dots(i+1)})g0^{i-1} - f(x_{n\dots(i+1)})0^i \\
\text{Compose: } f \xrightarrow{\beta} h \xrightarrow{\theta} \pi & \pi(x)_i = x_i - g + f(x_{n\dots(i+1)})g0^{i-1} \\
\text{Compose: } f \xrightarrow{\rho} h \xrightarrow{\theta} \pi & \pi(x)_i = x_i - g + f(0^{i-1}gx_{n\dots(i+1)}) \\
\text{Compose: } \pi \xrightarrow{\theta^{-1}} h \xrightarrow{\alpha^{-1}} f & f(x) = \pi(x)_{R(x)} + f(R_0(x)) \\
\text{Compose: } \pi \xrightarrow{\theta^{-1}} h \xrightarrow{\beta^{-1}} f & f(x) = \pi(x)_{R(x)} \\
\text{Compose: } \pi \xrightarrow{\theta^{-1}} h \xrightarrow{\rho^{-1}} f & f(x) = \pi(x_{(L(x)-1)\dots 1}x_{L(x)}0^{n-L(x)})_{n+1-L(x)} \\
\text{Compose: } \pi \xrightarrow{\theta^{-1}} h \xrightarrow{\lambda} h \xrightarrow{\theta} \pi & \pi^{-1}(x)_i = -\pi(\pi^{-1}(x)_{n\dots(i+1)})[-x_i]0^{i-1}_i
\end{array}$$

$$\text{Binary Operator: } h = h_1 * h_2 \quad h(s) = h_2(s) + h_1(\left(\prod_{j=|s|}^1 [s_j - h_2(s_{|s|\dots(j+1)})]\right))$$

Bijections to efficiently compute functions in other representations of \mathcal{F} .

$$\begin{array}{ll}
\text{Compose: } f \xrightarrow{\alpha} h \xrightarrow{\rho^{-1}} f & f'(x) = f(x_{(L(x)-1)\dots 1}x_{L(x)}0^{n-L(x)}) - f(x_{(L(x)-1)\dots 1}0^{n+1-L(x)}) \\
\text{Compose: } f \xrightarrow{\rho} h \xrightarrow{\alpha^{-1}} f & f'(x) = f(0^{R(x)-1}x_{R(x)}x_{n\dots(R(x)+1)}) + f'(R_0(x)) \\
\text{Compose: } f \xrightarrow{\alpha} h \xrightarrow{\beta^{-1}} f & f'(x) = f(x) - f(R_0(x)) \\
\text{Compose: } f \xrightarrow{\beta} h \xrightarrow{\alpha^{-1}} f & f'(x) = f(x) + f'(R_0(x)) \\
\text{Compose: } f \xrightarrow{\beta} h \xrightarrow{\rho^{-1}} f & f'(x) = f(x_{(L(x)-1)\dots 1}x_{L(x)}0^{n-L(x)}) \\
\text{Compose: } f \xrightarrow{\rho} h \xrightarrow{\beta^{-1}} f & f'(x) = f(0^{R(x)-1}x_{R(x)}x_{n\dots(R(x)+1)})
\end{array}$$

$R_0(x)$ is the mapping which sets the element $x_{R(x)}$ to 0.

APPENDIX B: Mathematica Code for Composition of Bijections

```
(* Code for symbolically composing various
bijections
*) (* Leemon Baird, 9 Jul 05 *) (* n is the length of the main
vectors we use.
vect[f, k] is a vector where element i is f[i], with i = 1 on the \
right, i = k on the left.
cat[a, b] is the concatenation of two vectors.
cat[a, b, c] is the concatenation of three vectors.
sub[v, i] is the ith element of
vector v (the rightmost is element 1).
sub[v, i, j] is the subvector formed by
elements i down to j of v. Empty if j > i.
len[v] is the length of vector v.
inverse[g] is the additive inverse of g in the group.
add[g1, g2] is the sum in the group (not necessarily commutative).
inv[g] is the additive inverse of g in the group.
sumfor[i, cond, exp] is the sum of exp[i] for all i where cond[i] \
is true.
compose[{b1, b2,
b3, ...}] returns the composition of bijections b1[b2[b3[ ...]]].
inverseBijection[b] returns the inverse of bijection b.
*)
showAllTeXcolumns := True; (*if False, this hides the first three \
columns in the TeX output (the text, forall g, f(x))*
Off[Function::flpar]; (*fixes bug in Mathematica*)
Off[General::spell1];
(*the new function fun[] is just like Function except that it always
\ evaluates its expressions immediately,
and it doesn't have the major Mathematica bug that Function suffers from :
Function[y, Function[z, f[y, z]]][z$] === > Function[z$, f[z$, \
z$]] BAD
Function[y, Function[z, f[y, z]]][z] === > Function[z$, f[z, \
z$]] CORRECT
fun[y, fun[z, f[y, z]]][z$] === > fun[z, f[z$, z]]
CORRECT
fun[y, fun[z, f[y, z]]][z] === > fun[v$27, f[z, \
v$27]] CORRECT
The functions replaceFree[], replaceBound[], and freeVars[] are \
just there to help define fun[].)
*)
(*this rule ensures that function application is immediately
evaluated
whenever possible*)
(*fun[i_, exp_][val_] := Block[{},
Print["#####" <> ToString[i] <> "#####" <> ToString[exp] <>
"#####" <> ToString[val] <> "#####"]; Block[{ans =
```

```

        replaceBound[exp, Append[freeVars[val], i]] /. {i -> val}},
        Print[ToString[ans] <> "####"]; ans]];*)
fun[i_, exp_][val_] := replaceBound[exp,
    Append[freeVars[val], i]] /. {i -> val};
(*replace each unbound occurrence of variable old with variable new
in
    expression exp.*)
(*a variable x is bound in exp if there is a fun[x, exp]*)
replaceFree[f_[x___],
    old_, new_] := Apply[f, Map[
    replaceFree[#, old, new] &, {x}]]; (*recurse down any structure,
    including Lists and functions, but not Sequences*)
replaceFree[x_, old_, new_] := x;
    (*stop recursing at the leaves of the structure*)
replaceFree[old_, old_, new_] := new;
replaceFree[fun[old_, exp_], old_, new_] := fun[old, exp];
(*create a unique new symbol like v$96 with
    Unique, then remove the dollar sign*)
gensym[s_] := Symbol[StringReplace[SymbolName[Unique[s]], {"$" -> ""}]];
(*Given a string
    containing gensymed variables starting with "v" or "j", \
replace them with better names*)improveVariables[str_] := Block[{badVars, \
usedVars, goodVars, newVars, ans},
    badVars = Union[StringCases[str, RegularExpression["(v|j)\d+"]]];
    usedVars = Union[StringCases[str, RegularExpression["\w+"]]];
    goodVars = {"j", "k", "a", "
b", "c", "d", "e", "
f", "g", "h
", "m", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"};
    newVars = Sort[Complement[goodVars,
    usedVars], (Position[goodVars, #1][[1, 1]] <
    Position[goodVars, #2][[1, 1]]) &][[Table[i, {i, 1, \
Length[badVars]}]]];
    ans = StringReplace[str, MapThread[(#1 -> #2) &, {badVars, newVars}]];
    ans];
(*replaceBound[exp, vars] will replace each bound variable
on the vars list with a unique gensym everywhere it occurs in that \
scope*)
replaceBound[f_[x___], vars_] := Apply[f, Map[
    replaceBound[#, vars] &, {x}]]; (*recurse down any structure, \
including Lists and functions, but not Sequences*)
replaceBound[
    exp_, vars_] := exp;
    (*stop \
    recursing at the leaves of the structure*)
replaceBound[fun[i_, exp_], vars_] := Block[{newVar},
    If[MemberQ[vars, i], newVar = gensym[v];
    fun[newVar, replaceBound[replaceFree[exp, i, newVar], vars]],
    fun[i, replaceBound[exp, vars]]];];

```

```

(*freeVars[exp] returns a list of variables that are not
   bound in exp.Duplicates are
   removed.An x is ignored in e if it's in a fun[x, e].*)
freeVars[f_[x_]] := Union[Apply[Join, Map[
   freeVars, {x}]]];      (*recurse down any
   structure, including Lists and functions, but not Sequences*)
freeVars[x_] := {};      (*stop
   recursing at the leaves of the structure*)
freeVars[x_Symbol] := {x};
freeVars[fun[i_, e_]] := DeleteCases[freeVars[e], i];

(*return the bijection b1[b2[]] found by composing bijections b1 and b2, or \
b1[b2[b3[]]] if it's passed {b1, b2, b3}, etc.*)
(*NOTE : it is erroneous to
   use this if both b2 and
   inverseBijection[b2] are recursively defined. At least one must be \
nonrecursive.*)
compose[{b1_, b2_}] := b1[#1, b2[inverseBijection[b1][ERROR, #1], #2]] &;
compose[{b_, b1_, b2_}] := compose[{b, compose[{b1, b2}]}];

(*define the bijections*)
bijHGamma [_, h_] := fun[x,
vect[fun[i, h[sub[x, n, i + 1]]], n]];
bijGammaH [_, gamma_] :=
fun[s, sub[gamma[cat[s, vect[fun[i, 0], n - len[s]]], n - len[s]]];
bijPiGamma [_, pi_] := fun[x, add[x, inv[pi[x]]];
bijGammaPi[_, gamma_] := fun[x, add[x, inv[gamma[x]]];
bij1FH [_, f_] := fun[s, add[add[g[s], inv[f[cat[s, g[
s], vect[fun[i, 0], n -
len[s] - 1]]]], f[cat[s, vect[fun[i, 0], n - len[s]]]]];
bij1HF [f_,
h_] := fun[x, add[add[sub[x, right[x]], inv[h[sub[x, n, right[
x] + 1]]], f[cat[sub[x, n, right[x] + 1], vect[fun[i, 0], right[
x]]]]];
bij2FH [_, f_] := fun[s, add[g[s], inv[f[cat[s, g[s], vect[
fun[i, 0], n - len[s] - 1]]]]];
bij2HF [_, h_] := fun[
x, add[sub[x, right[x]], inv[h[sub[x, n, right[x] + 1]]]];
bij3FH [_, f_] := fun[s, add[g[
s], inv[f[cat[vect[fun[i, 0], n - len[s] - 1], g[s], s]]]];
bij3HF [_, h_] := fun[x, add[sub[x, left[
x]], inv[h[sub[x, left[x] - 1, 1]]]];
bijHH [hi_, h_] :=
fun[s, inv[h[add[s, inv[vect[fun[i, hi[
sub[s, len[s], i + 1]], len[s]]]]]];
inverseBijection[bijHGamma] := bijGammaH;
inverseBijection[bijGammaH] := bijHGamma;
inverseBijection[bijPiGamma] := bijGammaPi;
inverseBijection[bijGammaPi] := bijPiGamma;
inverseBijection[bij1FH] := bij1HF;
inverseBijection[bij1HF] := bij1FH;

```



```

    x_] - 1, 1]] := leftReset[x]; (*define leftReset[x] which
        zeros out the leftmost nonzero digit*)
cat[sub[x_, a_, b_], sub[x_, c_], d_] := cat[sub[x, a, c], d] /;
    b == c + 1;
inv[add[a_,
    b_] := add[inv[a], inv[b]]; (*negation distributions over addition*)
inv[inv[a_]] := a; (*negation cancels*)
inv[0] := 0;
inv[vect[f_, k_] := vect[fun[i, inv[f[i]]], k];
len[vect[f_, k_] := k; len[add[_, vect[f_, k_], _]] := k;
len[sub[v_, i_, j_]] := i - j + 1; len[add[_, sub[v_, i_, j_],
    _]] := i - j + 1;
left[cat[vect[fun[i_, 0], _], g[z_], s_]] := 1 + len[s];
right[cat[s_, g[z_], vect[fun[i_, 0], k_]]] := k + 1;
sub[gamma[cat[sub[s_, n_, 1 + i_], vect[fun[j_,
    0], i_]]], i_] := sub[gamma[s], i]; (*gamma is strictly left dependent*)
sub[add[a_, b_], i_] := add[
    sub[a, i], sub[b, i]]; (*sub distributes over addition*)
sub[add[a_, b_], i_, j_] := add[sub[a, i, j], sub[b, i, j]]; (*
    sub distributes over addition*)
sub[inv[x_], k_] := inv[sub[x, k]]; (*sub distributes over inverse*)
sub[vect[f_, k_], j_] := f[j];
sub[vect[f_, a_], b_, c_] := Block[{var = gensym["v"]}, vect[fun[var, f[var + \
    c - 1]], b - c + 1]];
sub[cat[_ , vect[f_, k_]], k_] := f[k]; (*taking leftmost element of one of
    two concatenated vectors*)
sub[cat[_ , _ , vect[f_, k_]], k_] := f[k]; (*same for three vectors*)
sub[cat[_ , g[z_], s_], 1 + len[s_]] := g[z];
sub[cat[_ , _ , s_], len[s_], 1] := s;
sub[cat[s_, g[z_], vect[_ , -1 + k_]], k_] := g[z];
sub[cat[s_, g[z_], vect[_ , -1 + k_]], n_, 1 + k_] := s;
sub[cat[s_, g[z_], vect[_ , -1 +
    n_ - len[s_]]], n_, n_ - len[s_]] := cat[s, g[z]];
sub[cat[s_, vect[f_, n_ - len[s_]]], n_, n_ - len[s_] + 1] :=
    s; (*if sub takes first of two concatenated things*)
sub[sub[x_, a_, b_], c_] := sub[x, c + b - 1];
sub[sub[x_, a_, b_], c_, d_] := sub[x, c + b - 1, d + b - 1];
sub[cat[s_, vect[fun[i_, 0], k_]], t_] := sub[s, t -
    k];          (*ASSUMPTION : s_t is
    defined to be zero when t is nonpositive*)
sub[cat[s_, vect[fun[i_, 0], k_]], t_, v_] := sub[s, t - k, v - k];
sub[s_, len[s], 1] := s;
vect[fun[i_, sub[gamma[x_], i_]], n] := gamma[x] /;
    FreeQ[x, i]; (*gamma returns a vector of length n*)
vect[fun[j_, sub[newPi[x], j_ + k_]], a_] := sub[newPi[x], a + k, 1 + k];
(*****\
*****)

```

```

(** ** ** ***** ** ** rules to convert the output to TeX so it can be copied and p\
asted into a paper *****)
(*****\
*****\

totex[x_] := ToString[x]; totex[pi] \ := "\\pi "; totex[newPi] :=
"\\pi' "; totex[gamma] := "\\gamma "; totex[newGamma] := "\\gamma'";
totex[h1] := "h_1"; totex[h2] := "h_2"; totex[newF] := "f'";
totex[newH] := "h'"; totex[left] := "L"; totex[right] := \
"R";
totex[add[x_, y_]] := totex[x] <> If[StringTake[totex[y], 1] == "-", "", "+
"] <> totex[y];
totex[inv[x_]] := "-" <> totex[x];
totex[sub[x_, y_]] := totexParen[x] <> "_{" <> totex[y] <> "}";
totex[sub[x_, y_, z_]] := totexParen[x] <> "_{" <>
totexParen[y] <> "\\ldots " <> totexParen[z] <> "}";
totex[cat[x_, y_]] := totexParen[x] <> totexParen[y];
totex[cat[x_, inv[y_]]] :=
totexParen[x] <> "[-" <> totexParen[y] <>
"]"; (*concatenating x and - y shouldn't look like x - y, so
make it x[-y].*)
totex[cat[x_, y_, z_]] :=
totexParen[x] <> totexParen[y] <> totexParen[z];
totex[cat[x_, inv[y_], inv[z_]]] := totexParen[x] <> "[-" <> totexParen[y] <>
"][-" <> totexParen[z] <> "];
totex[cat[x_, inv[
y_, z_]] := totexParen[x] <> "[-" <>
totexParen[y] <> "]" <> totexParen[z];
totex[cat[x_, y_, inv[z_]]] :=
totexParen[x] <> totexParen[y] <> "[-" <> totexParen[z] <> "];
totex[f[x_]] := totex[f] <> "(" <> totex[x] <> " ";
totex[x_ + y_] := totex[x] <> "+" <> totex[y];
totex[x_ - y_] := totex[x] <> "-" <> totex[y];
totex[a_ + b_ - c_] := totex[a] <> "+" <> totex[b] <> "-" <> totex[c];
totex[-1 + x_] := totex[x] <> "-1";
totex[1 + x_] := totex[x] <> "+1";
totex[-1 + b_ + c_] := totex[b] <> "-1+" <> totex[c];
totex[-1 + b_ - c_] := totex[b] <> "-1-" <> totex[c];
totex[1 + b_ - c_] := totex[b] <> "+1-" <> totex[c];
totex[vect[fun[i_, 0], x_]] := "0^{ " <> totex[x] <> " }";
totex[vect[f_, k_]] := Block[{ctr =
gensym[j]}, "|_{ " <> ToString[ctr] <> "=" <>
totex[k] <> "}^1[" <> totex[f[ctr]] <> "];
totex[x_ ? y_] := totex[x] <> "\\neq " <> totex[y];
totex[len[x_]] := "|" <> totex[x] <> "|";
totex[rightReset[x_]] := "R_0(" <> totex[x] <> " ";
totex[leftReset[x_]] := "L_0(" <> totex[x] <> " ";
(*totex[Sum[exp_, {i_, a_, b_}]] := "\\sum_{ " <> totex[
i] <> "=" <> totex[a] <> "}^{ " <> totex[b] <> " }[" <> totex[exp] <> "]; \

```

```

(*this isn't used, currently*)
(*totex[sumfor[i_, cond_, exp_add]] := "\\sum{" <> totex[i] <> "\\ni " <> \
totex[cond[i]] <> "}" <> totex[exp] <> ""];*)
(*totex[sumfor[
  i_, cond_, exp_] := "\\sum{" <> totex[i] <> "\\ni
  " <> totex[cond[i]] <> "}" <> totex[exp];*)
(*put parentheses around complex expressions*)
totexParen[x_] := If[MemberQ[{Integer, Symbol, len, pi, f, h, gamma, cat,
  vect, sub, newPi, newGamma, newF, newH, x,
  s}, Head[x]], totex[x], "(" <> totex[x] <> ")];

(*output all the results to both the screen
  and the file "MathematicaOutput.tex"
  in the same directory as this Mathematica .nb file*)
strm = OpenWrite[StringJoin[
  Map[({# <> "\\") &, Rest[Last[Options[$FrontEnd, NotebooksMenu][[1, \
2]]][[2, 1, 1]]]] <> "MathematicaOutput.tex"];

(*output to screen and TeX file the result of applying
  bijection "bij" to function "oldF" to
  get function "newF[par].*)
(*the rules "subs" are then applied to the end result.*)
output[txt_, oldF_, newF_, par_, bij_, subs_] :=
  Block[{res, isVect = False, hasG = False, exp = bij[newF, \
oldF][par]},
  If[MatchQ[exp, vect[fun[i_, exp_], n]] ||
    MatchQ[exp, add[a___, vect[fun[i_, exp_], n], b___]], (*if the \
entire expression is a vector of n elements, possibly with other things \
added*)
    isVect = True;
    exp = sub[exp, i];
    exp = exp //. subs;
    If[Length[Position[exp, g]] ? 0, (*if there are any g'
s in the expression*)
      hasG = True;
    ];
    If[Length[Position[exp, ERROR]] ? 0,
      exp = "Can't compose recursive with inverse of recursive"];
    res = ToString[exp];
    Print[txt <> "      " <>
      ToString[newF] <> "(" <> ToString[par] <> ")"
      " <> If[isVect, "_i", ""] <> "= " <> ToString[res]];
    WriteString[strm,
      If[showAllTeXcolumns,
        StringReplace[ StringReplace[txt, " to " -> " \\rightarrow "],
          ":" -> ":%"]
          <> "&" <> "&" <> If[hasG, "\\textrm{ }\\forall g\\neq 0", ""]
          <> "&" <> totex[newF[par]]
          <> If[isVect, "_i", ""]

```

```

    <> "&",
    ""]
    <> "= " <> improveVariables[totex[exp]] <> "\\\\n";];

WriteString[strm,
  "\\begin{flalign*}\n"
  ]; (*header for the TeX file*)

output["Definition: h to \\gamma", h, gamma, x, bijHGamma, {}];
output["Definition: \\gamma to h", gamma, h, s, bijGammaH, {}];
output["Definition: \\pi to \\gamma", pi, gamma, x, bijPiGamma, {}];
output["Definition: \\gamma to \\pi", gamma, pi, x, bijGammaPi, {}];
output["Definition 1: f to h", f, h, s, bij1FH, {g[s] -> g}];
output["Definition 2: f to h", f, h, s, bij2FH, {g[s] -> g}];
output["Definition 3: f to h", f, h, s, bij3FH, {g[s] -> g}];
output["Definition 1: h to f", h, f, x, bij1HF, {}];
output["Definition 2: h to f", h, f, x, bij2HF, {}];
output["Definition 3: h to f", h, f, x, bij3HF, {}];
WriteString[strm, "\\\\n"];
output["Check: \\gamma to h to \\gamma", gamma, newGamma,
  x, compose[{bijHGamma, bijGammaH}], {}];
output["Check: h to \\gamma to h", h, newH, s, compose[{bijGammaH,
  bijHGamma}], {}];
output["Check: \\gamma to \\pi to \\gamma", gamma, newGamma, x, \
  compose[{bijPiGamma, bijGammaPi}], {}];
output["Check: \\pi to \\gamma to \\pi", pi, newPi, s,
  compose[{bijGammaPi, bijPiGamma}], {}];
output["Check 1: h to f to h", h, newH, s, compose[{bij1FH, bij1HF}], {}];
output["Check 2: h to f to h", h, newH, s, compose[{bij2FH, bij2HF}], {}];
output["Check 3: h to f to h", h, newH, s, compose[{bij3FH, bij3HF}], {}];
output["Check 1: f to h to f", f,
  newF, x, compose[{bij1HF, bij1FH}], {g[_] -> sub[x, right[x] ]}];
output["Check 2: f
  to h to f", f, newF, x, compose[{bij2HF, bij2FH}], {n -> len[x], \
  g[_] -> sub[x, right[x] ]}];
output["Check 3: f to h to f", f, newF, x, compose[{bij3HF,
  bij3FH}], {n -> len[x], g[_] -> sub[x, left[x] ]}];
output["Check inverse: h to h", h, newH, s, compose[{bijHH, bijHH}], {n -> \
  len[x], g[_] -> sub[x, left[x] ]}]; WriteString[strm, "\\\\n"];
output["Compose: \\pi to \\gamma to h", pi, h, s, compose[{
  bijGammaH, bijPiGamma}], {}];
output["Compose: h to \\gamma to \\pi", h, pi, x, compose[{bijGammaPi, \
  bijHGamma}], {}];
output["Compose 1: \\pi to \\gamma to h to f", pi, f, x, compose[{bij1HF,
  bijGammaH, bijPiGamma}], {}];
output["Compose 2: \\pi
  to \\gamma
  to h to f", pi, f, x, compose[{bij2HF, bijGammaH, bijPiGamma}], \
  {}];
output["Compose 3: \\pi to \\gamma to h to f", pi, f, x, compose[{bij3HF, \

```

```

bijGammaH, bijPiGamma}], {}];
output["Compose 1:
  f to h to \\gamma to \\pi", f, pi, x, compose[{bijGammaPi, bijHGamma,
  bij1FH}], {g[_] -> g}];
output["Compose 2: f to h to \\gamma to \\pi", f, pi, x, compose[{bijGammaPi, \\
bijHGamma, bij2FH}], {g[_] -> g}];
output["Compose 3: f to h to \\gamma to \\pi", f, pi, x,
  compose[{bijGammaPi, bijHGamma, bij3FH}], {g[_] -> g}];
output["Compose: \\pi to \\gamma to h to h to \\gamma to \\pi", pi, newPi, x, \\
compose[{bijGammaPi, bijHGamma, bijHH, bijGammaH, bijPiGamma}], {g[_] -> g}];
output["Compose 1: f to h to h
  to f", f, newF, x,
  compose[{bij1HF, bijHH, bij1FH}], {g[_] -> inv[sub[x, right[x]]]};
output["Compose 2: f to h to h to f", f, newF, x, compose[{bij2HF, bijHH, \\
bij2FH}], {g[_] -> inv[sub[x, right[x]]]};
output["Compose 3: f to h to h to f", f, newF, x, compose[{
  bij3HF, bijHH, bij3FH}], {g[_] -> sub[x, left[x]]};
output["Compose: f_1 to h
to f_3", f, newF, x, compose[{bij3HF, bij1FH}], {g[_] -> sub[x, left[x]]};
output["Compose: f_3 to h to f_1",
  f, newF, x, compose[{bij1HF, bij3FH}], {g[_] -> sub[x, right[x]]};
output["Compose: f_1 to h to f_2", f, newF, x, compose[{bij2HF, bij1FH}], {
  g[_] -> sub[x, right[x]]};
output["Compose: f_2 to h to
  f_1", f, newF, x, compose[{bij1HF, bij2FH}], {g[_] -> sub[x, right[x]]};
output["Compose: f_2 to h to f_3", f, newF, x, compose[{
  bij3HF, bij2FH}], {g[_] -> sub[x, left[x]]};
output["Compose:
  f_3 to h to f_2", f, newF, x, compose[{bij2HF, bij3FH}], {g[_] -> sub[x,
  right[x]]};
output["Compose 1: f to h to h to \\pi", f,
  pi, x, compose[{bijGammaPi, bijHGamma, bijHH, bij1FH}], {g[_] -> g};
output["Compose 2: f to h to h to \\pi", f, pi, x, compose[{bijGammaPi,
  bijHGamma, bijHH, bij2FH}], {g[_] -> g};
output["Compose 3:
  f to h to h
  to \\pi", f, pi, x,
  compose[{bijGammaPi, bijHGamma, bijHH, bij3FH}], {g[_] -> g};
output["Compose
1: \\pi to h
  to h to f", pi, f, x, compose[{bij1HF, bijHH, bijGammaH, bijPiGamma}], {
  g[_] -> g};
output["Compose 2: \\pi to
h to h to f",
  pi, f, x, compose[{bij2HF, bijHH, bijGammaH, bijPiGamma}], {g[_] -> g};
output["Compose 3: \\pi to h to h to f", pi, f, x, compose[{
  bij3HF, bijHH, bijGammaH, bijPiGamma}], {g[_] -> g};

pi2 = bijGammaPi[ERRORERROR, bijHGamma[ERRORERROR, h2]]; (*pi2[x]

```

```

    is the permutation*)
pi1 = bijGammaPi[ERRORERROR, bijHGamma[ERRORERROR, h1]];
pi3 = fun[x, pi1[pi2[x]]];
h3 = bijGammaH[ERRORERROR, bijPiGamma[ERRORERROR, pi3]];
h3old = bijGammaH[_ , bijPiGamma[_ ,
    bijGammaPi[_ , bijHGamma[_ , h1]][bijGammaPi[_ , bijHGamma[_ , h2]][x]]];
output["Isomorphism: h = h_1 * h_2", newH, h, s, h3 &, {}];
output["Derivative: f(x+v)-f(v) to \pi_2(x)", (f[# + v] - f[v]) &,
pi, x, compose[{bijGammaPi, bijHGamma, bij2FH}], {g[_] -> g}];
output["Derivative: \pi(x+v) to f_2(x)
    ", (pi[# + v]) &, f,
    x, compose[{bij2HF, bijGammaH, bijPiGamma}], {}];
WriteString[strm,
    "\end{flalign*}\n"
]; (*footer for the TeX file*)
Close[strm];

```