Parallel SSEP and a Casimir Element of $\mathfrak{so}_{2n}$

Mark Landry, Andrew Park

July 23, 2020

Abstract

Recent research has shown that one can obtain the generator of a Markov process through the choice of a Casimir element and representation of a chosen Lie algebra. In this paper we introduce the notion of a parallel symmetric simple exclusion process, a system made up of distinct SSEPs, in order to classify symmetric processes obtained from $\mathfrak{so}_{2n}$. These classifications involve a rigorously defined type of parallel SSEP by use of properties relating to ideas of class, mass, and mass-preserving transformations that allow concurrent moves elsewhere in the system.

1 Introduction

A Markov process is a continuous-time physical process with a discrete number of states where states jump to other states at random times and the probabilities depend only on the present state the process is in, not past states the process has been in. Generator matrices are a tool used often to encode the transition rates between states in a Markov process, and also have implications for probabilities regarding behavior of the process.

Recently, it has become more popular to study these stochastic processes through the Lie algebra interpretation of their generators; it has been shown that there is a connection between the two subjects that allows the use of algebraic methods in studying the properties of certain Markov processes.

One example of this method comes from the Lie algebra $\mathfrak{sl}_2$, which can be used to obtain the generator matrix of the simple symmetric exclusion process (SSEP). The SSEP particle system involves a lattice with two sites, each of which can be empty or contain a particle; a particle can jump to an empty neighboring site but is blocked from jumping if the site is occupied. Moreover, all jump rates are equal due to symmetry in the model.

The generator for the SSEP model can be obtained using a distinguished central element called the Casimir element and a representation of the universal enveloping algebra of $\mathfrak{sl}_2$; this method can be generalized to other Lie algebras to obtain the generators of other processes. Naturally, more complicated Lie algebras result in more complicated systems. For this paper, we aim to classify a symmetric Markov process obtained from $\mathfrak{so}_{2n}$ for $n > 1$.

2 Background

2.1 Generator Matrices and Probability in Markov Processes

First, we list a well-known proposition from the study of Markov processes that demonstrates one of the implications a generator matrix has for probabilities within its corresponding Markov process.

Proposition 1. *Let Q be a generator matrix, and let q_{xy} be the (x, y) entry of Q. Let T_x be the holding time at state x. If $q_{xx} \neq 0$, then $P(X_{T_X} = y | X_0 = x) = \frac{q_{xy}}{-q_{xx}}$.*

This proposition helps in understanding how states transition between each other in a Markov process, and is incredibly useful in describing a Markov process given a generator matrix.

2.2 $\mathfrak{so}_{2n}$ as a Lie Algebra

$\mathfrak{so}_{2n}$ is the Lie algebra of matrices of the form:

$$\mathfrak{so}_{2n}(\mathbb{C}) = \left\{ \begin{pmatrix} A & B \\ C & D \end{pmatrix} \middle| A, B, C, D \in \mathbb{C}^{n\times n}, A = -D^T, B = B^T, C = C^T \right\}$$

Let $E_{i,j}$ be the matrix with 1 in the (i, j) entry and 0 elsewhere. A Cartan subalgebra $\mathfrak{h}$ of $\mathfrak{so}_{2n}$ is the subalgebra of diagonal matrices of the form $H_i = E_{i,i} - E_{n+i,n+i}$.

Let L_i be defined as the linear map on the Cartan subalgebra where L_i maps H to its i-th diagonal entry. Then the roots and root vectors of $\mathfrak{so}_{2n}$ are defined for $i, j \leq n$ as:

- $X_{i,j} = E_{i,j} - E_{n+j,n+i}$ has root $L_i - L_j$
- $Y_{i,j} = E_{i,n+j} - E_{j,n+i}$ has root $L_i + L_j$
- $Z_{i,j} = E_{n+i,j} - E_{n+j,i}$ has root $-L_i - L_j$

The positive roots are $R^+ = \{L_i+L_j\}_{i<j} \cup \{L_i-L_j\}_{i<j}$, and the simple roots are $L_1-L_2, L_2-L_3, \ldots, L_{n-1}-L_n, L_{n-1}+L_n$. This implies that the rank of $\mathfrak{so}_{2n}$ is n, which corresponds to the number of H matrices. Additionally, in $\mathfrak{so}_{2n}$, the Killing Form can be defined using $B(X, Y) = (2n - 2)\,\mathrm{Tr}(XY)$.

3 Main Results

3.1 Analyzing the Generator Matrix from $\mathfrak{so}_{2n}$

3.1.1 Obtaining G_n

In this section, we describe the process of arriving at G_n, a generator matrix of a Markov process, from the Casimir element of a Lie algebra of the form $\mathfrak{so}_{2n}$ for $n \geq 2$.

First, note that the Cartan-Weyl Basis and corresponding dual basis for any $\mathfrak{so}_{2n}$ follows the same form. Let B be the Cartan Weyl Basis of $\mathfrak{so}_{2n}$ and let B' be the dual basis with respect to B. The elements of B and B' are the following:

For all $i \leq n$:

- $H_i \in B$ has dual $\frac{1}{4n-4} H_i \in B'$.

For all $i < j \leq n$:

- $X_{ij} \in B$ has dual $\frac{1}{4n-4} X_{ji} \in B'$.
- $X_{ji} \in B$ has dual $\frac{1}{4n-4} X_{ij}\ in B'$.
- $Y_{ij} \in B$ has dual $-\frac{1}{4n-4} Z_{ij} \in B'$.
- $Z_{ij} \in B$ has dual $-\frac{1}{4n-4} Y_{ij} \in B'$.

These counterparts can be calculated based on the Killing Form (H_i and X_{ij} products have trace 2, and Y_{ij} and Z_{ij} products have trace -2). Note that B' contains scaled versions of every element of B, showing that because B is a basis, B' is as well.

The Casimir element is $\Omega = \sum_i A_i A^i$ where each A_i is a unique element of the Cartan-Weyl basis and A^i is its counterpart in the dual basis. Let $\mathbb{C}^{2n}$ be the usual vector representation on $\mathfrak{so}_{2n}$, so $\rho_{\mathbb{C}^{2n}\otimes\mathbb{C}^{2n}}$ defines

a representation. Let $\rho = \rho_{\mathbb{C}^{2n}\otimes\mathbb{C}^{2n}}$. Thus, $\rho(\Omega) = \sum_i \rho(A_i)\rho(A^i)$ defines a $4n^2 \times 4n^2$ matrix. Let $\rho_{ij}(\Omega)$, $1 \leq i, j \leq 2n$ denote its entries.

Let C be a $4n^2 \times 4n^2$ diagonal matrix with entries c_i such that $c_i = \sum_j \rho_{ij}(\Omega)$. Let G_n be the matrix found by negating rows of $\rho(\Omega) - C$ to force each diagonal entry to be non-positive. This leads to a generator matrix with all entries finite.

Python code demonstrating the steps of this process to build G_n from n is located in Appendix A.

Remark. *Previous research on similar Lie algebras has used $C = kI$ for scalar k and then used conjugation to arrive at a generator matrix [2] as the matrix C. In $\mathfrak{so}_4$, this process and the definition of C described above arrive at the same result, and for $n > 2$, conjugation fails to arrive at a generator matrix where all entries are finite.*

3.1.2 Properties of G_n

In this section, we outline properties of G_n that will be helpful in analyzing the states of the Markov process G_n describes.

Lemma 1. *The representation of the Casimir element $\rho(\Omega)$, a $4n^2 \times 4n^2$ matrix, can be written as a $2n \times 2n$ matrix of blocks of size $2n \times 2n$ in block form as:*

$$\rho(\Omega) = \frac{1}{2n-2}\left(\begin{array}{ccccc|ccccc} D_1 & X_{21} & X_{31} & \cdots & X_{n,1} & 0 & -Z_{12} & -Z_{13} & \cdots & -Z_{1,n} \\ X_{12} & D_2 & X_{32} & \cdots & X_{n,2} & Z_{12} & 0 & -Z_{23} & \cdots & -Z_{2,n} \\ X_{13} & X_{23} & D_3 & \cdots & X_{n,3} & Z_{13} & Z_{23} & 0 & \cdots & -Z_{3,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ X_{1,n} & X_{2,n} & X_{3,n} & \cdots & D_n & Z_{1,n} & Z_{2,n} & Z_{3,n} & \cdots & 0 \\ \hline 0 & -Y_{12} & -Y_{13} & \cdots & -Y_{1,n} & D_{n+1} & -X_{12} & -X_{13} & \cdots & -X_{1,n} \\ Y_{12} & 0 & -Y_{23} & \cdots & -Y_{2,n} & -X_{21} & D_{n+2} & -X_{23} & \cdots & -X_{2,n} \\ Y_{13} & Y_{23} & 0 & \cdots & -Y_{3,n} & -X_{31} & -X_{32} & D_{n+3} & \cdots & -X_{3,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ Y_{1,n} & Y_{2,n} & Y_{3,n} & \cdots & 0 & -X_{n,1} & -X_{n,2} & -X_{n,3} & \cdots & D_{2n} \end{array}\right)$$

where $D_i = (2n-1)I + H_i$ and $D_{n+i} = (2n-1)I - H_i$ for $i \leq n$.

Proof. We can partition the Cartan Weyl basis for $\mathfrak{so}_{2n}$ into disjoint subsets according to pairs (i, j) for $i < j \leq n$, keeping the Cartan subalgebra generators (H_i matrices) separate. Each of the subsets will, for a pair (i, j), contain X_{ij}, X_{ji}, Y_{ij}, Z_{ij}.

Suppose we take an arbitrary part of the partitioned basis, i.e. choose (i, j) such that $i < j \leq n$. We will analyze the contribution this part (specifically, the X_{ij}, X_{ji}, Y_{ij}, and Z_{ij}) makes to the Casimir representation.

Each of these matrices can have their representation written in block form, which places the matrix on each of the diagonals and places I's, the identity matrix, in the locations indicated by the matrix. For example, in $\mathfrak{so}_4$:

$$\rho(X_{12}) = X_{12} \otimes I_4 + I_4 \otimes X_{12} = \begin{pmatrix} X_{12} & I & 0 & 0 \\ 0 & X_{12} & 0 & 0 \\ 0 & 0 & X_{12} & 0 \\ 0 & 0 & -I & X_{12} \end{pmatrix}.$$

Take $\rho(X_{ij})$ and $\rho(X_{ji})$ using this form. Ignoring the $\frac{1}{4n-4}$ scaling, we want to analyze $\rho(X_{ij})\rho(X_{ji}) + \rho(X_{ji})\rho(X_{ij})$. In order to do so, we will analyze a 6×6 block matrix which captures the contributions to the

$i, j, k, n+i, n+j, n+k$ block rows and columns, where $k \leq n$ such that $k \neq i, j$. Letting $P_{ij} = X_{ij}X_{ji} + X_{ji}X_{ij}$, this results in the following matrix:

$$\rho(X_{ij})\rho(X_{ji}) + \rho(X_{ji})\rho(X_{ij}) = \begin{pmatrix} P_{ij}+I & 2X_{ji} & 0 & 0 & 0 & 0 \\ 2X_{ij} & P_{ij}+I & 0 & 0 & 0 & 0 \\ 0 & 0 & P_{ij} & 0 & 0 & 0 \\ 0 & 0 & 0 & P_{ij}+I & -2X_{ij} & 0 \\ 0 & 0 & 0 & -2X_{ji} & P_{ij}+I & 0 \\ 0 & 0 & 0 & 0 & 0 & P_{ij} \end{pmatrix}.$$

Notice that P_{ij} is a diagonal matrix with 1 on the $i, j, n+i, n+j$ diagonals and 0 elsewhere. The block form of the identity matrix also has an I added to the P_{ij} at the $i, j, n+i, n+j$ rows. This means that, in the full $\rho(\Omega)$ matrix, the X terms contribute $2(n-1)I$ to each diagonal term. Notice also that X_{ij} and X_{ji} appear in blocks of $\rho(\Omega)$ at locations determined by $i, j, n+i, n+j$.

Likewise, take $\rho(Y_{ij})$ and $\rho(Z_{ij})$ using this form. Note that the coefficient of the dual basis elements will be negative ($\frac{-1}{4n-4}$), meaning these matrices will be negated before being included in the Casimir sum. Analyzing this matrix the same way as the X's, and letting R_{ij} be $Y_{ij}Z_{ij} + Z_{ij}Y_{ij}$, we get the result:

$$\rho(Y_{ij})\rho(Z_{ij}) + \rho(Z_{ij})\rho(Y_{ij}) = \begin{pmatrix} R_{ij}-I & 0 & 0 & 0 & 2Z_{ij} & 0 \\ 0 & R_{ij}-I & 0 & -2Z_{ij} & 0 & 0 \\ 0 & 0 & R_{ij} & 0 & 0 & 0 \\ 0 & 2Y_{ij} & 0 & R_{ij}-I & 0 & 0 \\ -2Y_{ij} & 0 & 0 & 0 & R_{ij}-I & 0 \\ 0 & 0 & 0 & 0 & 0 & R_{ij} \end{pmatrix}.$$

Notice that R_{ij} is a diagonal matrix with -1 on the $i, j, n+i, n+j$ diagonals and 0 elsewhere. The block form of the identity matrix also has an I subtracted from R_{ij} at the $i, j, n+i, n+j$ rows. This means that, in the full $\rho(\Omega)$ matrix, since this matrix is negated, the Y, Z terms contribute $2(n-1)I$ to each diagonal term. Notice also that Y_{ij} and Z_{ji} appear in locations determined by $i, j, n+i, n+j$, and again note that these will be negated in the final result.

In summary, we have shown that from pairs of (i, j) in the X, Y, Z elements of the basis, we generate each off-diagonal block based on one root vector and an equal value of $4(n-1)I$ in each diagonal block.

To fully analyze a block form, we must also analyze the contributions of the H matrices to $\rho(\Omega)$. However, these matrices are diagonal, so their representations will also be diagonal and they will only contribute to the diagonal blocks. We can define a block representation of H_i, which puts H_i in each diagonal block, adds I in the i-th diagonal block, and subtracts I in the $n+i$-th diagonal block.

Define a 4×4 matrix indexed by $i, j, n+i, n+j$ such that $j \neq i$ and $j \leq n$. The result is the following matrix (up to scaling):

$$\rho(H_i)\rho(H_i) = \begin{pmatrix} H_i^2 + 2H_i + I & 0 & 0 & 0 \\ 0 & H_i^2 & 0 & 0 \\ 0 & 0 & H_i^2 - 2H_i + I & 0 \\ 0 & 0 & 0 & H_i^2 \end{pmatrix}.$$

Adding this up for all i shows that this contributes differently to each diagonal. Let d_i be the i-th diagonal block of $\sum_i \rho(H_i)\rho(H_i)$. Then $d_i = 2I + 2H_i$ and $d_{n+i} = 2I - 2H_i$ for $i \leq n$.

When we put all of these pieces together, add the scaling back in (putting $\frac{1}{4n-4}$ on the outside of the matrix, as this was the coefficient in every case), and pull a constant of 2 out of the matrix (leading to $\frac{1}{2n-2}$ as the coefficient), we get the block form in the lemma. □

Lemma 2. *The G_n matrix is the generator of a Markov process.*

Proof. By the procedure used to construct the G_n from $\rho(\Omega)$, it immediately follows that all of the rows (in the normal sense) of the resulting matrix sum to 0 and their diagonals are non-positive. Therefore it suffices to show that all off-diagonal entries of are non-negative.

Using the block form of $\rho(\Omega)$, we see that the set of rows of $\rho(\Omega) - C$ with negative elements is given by $S = \{n+1, n+1+1(2n+1), n+1+2(2n+1), \ldots, 2n^2, 2n^2+1, 2n^2+1+1(2n+1), \ldots, 4n^2-n\}$. This constitutes the $n+1$ row of the 1st block, the $n+2$ row of the 2nd block., etc., up to the $2n$ row of the n-th block, and then the 1st row of the $n+1$-th block, the 2nd row of the $n+2$-th block, etc. We also notice that each of these rows is given a negative value by every block that is not on the diagonal or the 0 diagonals of the upper right and bottom left. This means that each of these rows contains $2n-2$ off diagonal negative elements, all of which will be -1, because they come from X, Y, Z matrices.

We want to show that the diagonal value in each row of S is equal to $2n-2$, which would imply that these rows already sum to 0 and thus will solely be negated, not adjusted by a constant. This is true because, in the D_i's, $2n-2$ will occur at the $n+1$ row, and then every $2n+1$ rows up to $2n^2$. In the D_{n+i}'s, $2n-2$ will occur at the $2n^2+1$ row and then every $2n+1$ rows after that up to $4n^2-n$. Thus, the rows indexed by S will have a 0 on that diagonal of C and will be the rows negated at the end.

Note that all other rows of C will have non-negative diagonals and non-negative off-diagonals, which will force the diagonals to become negative when C is subtracted and thus the off-diagonals will remain positive.

Hence, all off-diagonal entries are non-negative, and thus the matrix G_n is a generator matrix of a Markov Process. □

Past research has shown that the Markov process this generator matrix describes will involve a particle system with 2 sites, which can be expanded to N sites using a certain formula on the generator matrix. [1]

Because G_n is a $4n^2 \times 4n^2$ matrix, we can see that G_n will represent a Markov process with $4n^2$ states.

Lemma 3. *All non-zero off-diagonal entries of G_n are equal.*

Proof. It follows from block form that all off-diagonal entries of G_n occur in the off-diagonal blocks, which are determined solely by the X, Y, Z matrices. However, the X, Y, Z matrices contain entries of only $1, 0, -1$. Since all off-diagonal elements are non-negative, this implies that all nonzero off-diagonal entries of the matrix are the same and, when scaling is taken into account, equal to $\frac{1}{2n-2}$. □

This lemma is useful because it means that the diagonals will follow the same pattern as the rows in regards to what states they represent, and since all off-diagonal entries are equal, the diagonal of a row captures what kind of state it represents in a Markov process.

3.1.3 Expected Properties of the Markov Process from G_n

In this section, we will use the properties of G_n to show what states the Markov process that G_n describes would need to include.

Lemma 4. *The Markov process with generator G_n has $2n$ absorbing states.*

Proof. First, note that an absorbing state is indicated in a generator matrix by a row where all entries are 0.

Notice from block form that the first row of G_n for any n has 0's for all off-diagonal elements (the first row of all X_{j1} and Z_{1j} is all 0's). This means that the first diagonal entry of C will be equal to the first diagonal entry of $\rho(\Omega)$, which is $\frac{2n}{2n-2}$ and this will return an entire row of 0's.

Moreover, because every off-diagonal entry is positive and all nonzero off-diagonals are equal, this implies that every time the value $\frac{2n}{2n-2}$ recurs in the diagonal, it implies an absorbing state. By the structure of the D matrices, this occurs once in every D block, of which there are $2n$ total, implying $2n$ rows of all 0's. In a generator matrix, these types of rows imply absorbing states, showing that the Markov Process represented by this generator matrix has $2n$ absorbing states. □

Definition 1. *We define a* ***maximal choice row*** *as a row in which no other rows have a greater number of nonzero off-diagonal elements. The set of all maximal choice rows is called the* ***maximal choice set****.*

Lemma 5. *The generator G_n has $2n$ maximal choice rows, each of which have $2n-2$ non-zero off-diagonal entries at other maximal choice rows.*

Proof. Let $S = \{n+1, n+1+1(2n+1), n+1+2(2n+1), \ldots, 2n^2, 2n^2+1, 2n^2+1+1(2n+1), \ldots, 4n^2-n\}$. We want to show S is the maximal choice set. Recall from Lemma 2 that each row of S has $2n-2$ off-diagonal elements. Because each block of block form, as given by a root vector, has at most 1 entry in a row, and each block row has a 0 block, $2n-2$ is the maximal number of choices a row can have. This means that every element of S is a maximal choice row.

Moreover, the structure of the block form matrix shows that every other row has less than $2n-2$ off-diagonal entries, implying that S is the maximal choice set. Because S has cardinality $2n$, there are $2n$ maximal choice rows.

By a similar process to finding the rows of G_n with negative elements, thus populating S, and taking advantage of symmetry within the matrix, we can see that negative off-diagonals also only occur in columns indicated by elements of S. This implies that rows of S only communicate to other rows of S, making S a communicating class. Moreover, since each root vector has only one negative off-diagonal and using the structure of the block matrix, each row in S fails to immediately reach a unique other row (based on the location of the 0 matrix), but all $2n-2$ other states in S can be reached. □

Definition 2. *We define a* ***pairwise row*** *as a row with exactly one nonzero off-diagonal entry. If pairwise row r has its nonzero off-diagonal entry at column s, and s is a pairwise row with nonzero off-diagonal entry at column r, rows r and s are called* ***pairwise states****.*

Lemma 6. *Any row in G_n either represents an absorbing state, or is a maximal choice row, or is a pairwise state.*

Proof. Let r be a row of G_n that does not represent absorbing state and is not maximal choice row. Because r does not represent an absorbing state, it must have at least one nonzero off-diagonal element. All of the negative off-diagonal elements of $\rho(\Omega)$ are used in maximal choice rows, and thus we wish to study the positive off-diagonal elements of $\rho(\Omega)$.

First, we show that r is a pairwise row. If r is in the top half of G_n, r is in a block row s made up of three types of matrices: X_{ks} for $k \leq n$, Z_{ts} for $t < s$, and $-Z_{sq}$ for $q > s$. Each X_{ks} matrix will have its positive element in row k, each Z_{ts} will have its positive element in row $n+t$, and each $-Z_{sq}$ will have its positive element in row $n+q$. This forms a partition of all of the non-absorbing and non-maximal rows in the top half of the matrix, so r only contains one off-diagonal positive element. Similarly, we can show that if r is in the bottom half of the block form matrix, r only contains one off-diagonal positive element. Thus, r is a pairwise row.

Next, we know that for each r, its one off-diagonal element is in a block determined by a root vector. We will show that this root vector implies a corresponding pairwise row r' that makes r, r' pairwise states. This can be done in 4 cases:

1. If r is a row that has its positive entry in the block X_{ji}, then it has positive entry at matrix coordinates $(j+(i-1)2n, i+(j-1)2n)$. The block X_{ij} has its positive entry at matrix coordinates $(i+(j-1)2n, j+(i-1)2n)$. If r' is the row that has its positive entry in block X_{ij}, then r and r' are pairwise states.

2. If r is a row that has its positive entry in block $-X_{ij}$, then it has positive entry at matrix coordinates $(2n^2+(i-1)2n+n+i, 2n^2+(j-1)2n+n+j)$. The block $-X_{ji}$ has its positive entry at matrix coordinates $(2n^2+(j-1)2n+n+j, 2n^2+(i-1)2n+n+i)$. If r' is the row that has its pairwise entry in block $-X_{ji}$, then r and r' are pairwise states.

3. If r is a row that has its positive entry in block Y_{ij}, then it has positive entry at matrix coordinates $(2n^2 + (j-1)2n + i, (i-1)2n + n + j)$. The block $-Z_{ij}$ has its positive entry at matrix coordinates $((i-1)2n + n + j, 2n^2 + (j-1)2n + i)$. If r' is the row that has its pairwise entry in block $-Z_{ij}$, then r and r' are pairwise states.

4. If r is a row that has its positive entry in block Z_{ij}, then it has its positive entry at matrix coordinates $((j-1)2n + n + i, 2n^2 + (i-1)2n + j)$. $-Y_{ij}$ as its positive entry at matrix coordinates $(2n^2 + (i-1)2n + j, (j-1)2n + n + i)$. If r' is the row that has its pairwise entry in $-Y_{ij}$, then r and r' are pairwise states.

Hence, we see that every row that is not absorbing or maximal choice has exactly one positive off-diagonal and communicates to and from exactly one other row. □

We have seen that the rows of G_n imply a Markov Process that can be split up into completely independent communicating classes, i.e. no state in a communicating class can reach a state in another communicating class. We have $2n$ communicating classes with 1 row (absorbing states), 1 communicating class with $2n$ rows such that each row can reach all others but 1 uniquely (maximal choice rows), and $2n^2 - 2n$ communicating classes of pairwise states, totaling $4n^2$ states. We will now turn our focus to the Type-m Parallel SSEP and show that these criteria apply to this particle system.

3.2 The Particle System

Definition 3. *A **Parallel SSEP with N sites** is a system that has two separate 1-dimensional SSEPs with $N \geq 2$ sites each. Each site can either be empty or have a particle on it, and while particles can interact with neighboring sites on the same lattice, they cannot jump to the other lattice. A Parallel SSEP with 2 sites will be referred to as a **Basic Parallel SSEP**.*

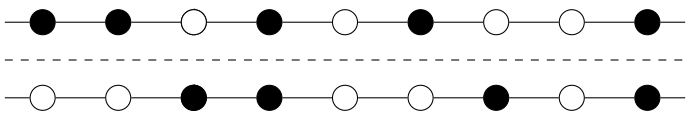

Figure 1: A possible configuration of a Parallel SSEP with 9 sites

Definition 4. *A **Type-m Basic Parallel SSEP** is a Basic Parallel SSEP where the lower lattice allows particles of mass 1, the upper lattice allows particles of mass $\omega \in \{\frac{1}{m}, \frac{2}{m}, \ldots, \frac{m-1}{m}, 1\}$, and the following properties hold:*

- ***Mass Order Property**: A particle can only move if no lighter particles can move.*
- ***Balance Property**: A set of balanced states exists in which the two lattices each have mass 1 and particles are able to undergo fusion and fission, defined respectively as donating mass to or taking mass from a neighboring site. These mass-preserving processes allow, but do not force, a concurrent move in the lower lattice.*
- ***Class Property**: A particle in an unbalanced state is able to switch places with neighboring particles of higher mass.*

Lemma 7. *A Type-m Basic Parallel SSEP has $4(m+1)^2$ states.*

Proof. By the definition of Type-m Basic Parallel SSEP, the particle system contains 2 parallel latices with two sites each. The top lattice consists of 2 sites with $m+1$ configurations each (m types of particles or empty). Thus, the top lattice has $(m+1)^2$ configurations.

Similarly, because the bottom lattice consists of 2 sites with 2 states each, there are 4 possible configurations of the bottom lattice. Since all possible combinations of a configuration of the top lattice and a configuration of the bottom lattice are valid states, there are $4(m+1)^2$ states. □

Lemma 8. *A Type-m Basic Parallel SSEP has $2(m+1)$ absorbing states.*

Proof. Absorbing states are reached by having each lattice occupied by either a pair of identical particles or a pair of empty sites. There are $m+1$ ways to accomplish this on the top lattice, and independently 2 ways to accomplish this on the bottom lattice. Thus, there are $2(m+1)$ absorbing states. □

Definition 5. *We define **maximum-choice states** as the states such that no other possible states in the system have more **choices**, states that can be immediately reached.*

Lemma 9. *A Type-m Basic Parallel SSEP has $2(m+1)$ maximum-choice states. Each of these states have $2m$ choices, each of which is another maximum-choice state.*

Proof. First we will prove that maximum-choice states only occur when the top and bottom lattice are equal at mass 1 each, which is a balanced state. Begin by noting that if the lattices are equal at mass 1, the state necessarily has more than 1 choice. Assume then that the top and bottom lattice are not equal, for if they are equal at mass 2 we are in an absorbing state. Then, if there is a movable particle of lowest mass it will only have 1 choice of movement, thus an unbalanced state has at most 1 choice. Since balanced states are our only maximum-choice states, we then know that unbalanced states can not be maximum-choice.

Thus, maximum-choice states occur only when the mass of the top lattice and the mass of the bottom lattice are equal at 1. This requires the two sites on the top lattice to have mass summing to 1, and there are $m+1$ ways to arrange this. Independently, one particle of mass 1 must be on one of the 2 sites on the bottom lattice. Thus, there are $2(m+1)$ maximum-choice states.

To prove the second part, we let our system be in a maximum-choice state. We first note that all state changes will preserve the mass of both the top and bottom lattice, so a maximum-choice state can only change into other maximum-choice states. We then have at most $2(m+1)$ choices, and since a state can't change to itself we have at most $2m+1$ choices. We note that we have one particle of mass 1 on the bottom. There are two cases a balanced state can follow:

1. If our top lattice also has just one particle of mass 1 on one of its sites, then the state where both particles switch sites simultaneously is not a possible choice, but all other balanced states are reachable.
2. Alternatively, if have two particles of mass less than 1 on the top lattice which add up to mass 1, then the state where the top lattice's configuration stays the same and the bottom particle switches sites is not a possible choice, but all other balanced states can be reached.

We then have $2m$ remaining choices, which are easily shown to all be reachable from the current state.

□

Lemma 10. *A state in the Type-m Basic Parallel SSEP that is not absorbing or balanced must be pairwise.*

Proof. Let X be a state that is not absorbing or balanced. X cannot be the empty state, so it must have a movable particle of lowest mass. This particle will switch sites, resulting in the state X'. However, the movable particle of lowest mass from X is still the movable particle of lowest mass in X', and X' will then switch back to X. Thus, X is a pairwise state. □

Theorem 1. *Let $m = n-1$. The generator matrix of a Type-m Basic Parallel SSEP is exactly G_n.*

Proof. The previous lemmas demonstrate that the states implied in a Markov Process by the generator matrix, derived from a representation of a Casimir Element of $\mathfrak{so}_{2n}$, are identical to the states in a Type-m Basic Parallel SSEP. The $2n$ rows of all 0 in the matrix correspond to $2n$ absorbing states in the system, the $2n$ maximal choice rows correspond to balanced, or maximum-choice, states, and everything else in both is split up into pairwise states. Hence, we conclude by state analysis that G_n as a generator matrix represents Type-m Basic Parallel SSEP. □

3.2.1 An Example: $\mathfrak{so}_4$

We analyze a specific case of $\mathfrak{so}_{2n}$, specifically the $n = 2$ case. Note that $\mathfrak{so}_4 \cong \mathfrak{sl}_2 \oplus \mathfrak{sl}_2$, lending support to the Parallel SSEP model.

The Cartan-Weyl basis for $\mathfrak{so}_4$ is $B = \{H_1, H_2, X_{12}, X_{21}, Y_{12}, Z_{12}\}$. The Killing Form calculation implies that the dual basis for the Cartan-Weyl Basis is $B' = \{\frac{1}{4}H_1, \frac{1}{4}H_2, \frac{1}{4}X_{21}, \frac{1}{4}X_{12}, -\frac{1}{4}Z_{12}, -\frac{1}{4}Y_{12}\}$, corresponding to the order of the original Cartan-Weyl basis. Using the process described above, we arrive at a generator matrix for a 16-state Markov process.

The 16×16 generator matrix that results from $\mathfrak{so}_4$ is the following matrix:

$$G_2 = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & -1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & -1 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

As the generator of a Markov process, this matrix implies that there are 4 absorbing states (Rows 1, 6, 11,and 16) and a communicating class with 4 maximum-choice rows with 2 choices each (Rows 3, 8, 9, and 14). The other states are divided into communicating classes of a pair that switches back and forth (Rows 2 & 5, 4 & 13, 7 & 10, and 12 & 15). Note that G_2 can be rewritten as a block matrix based on the communicating classes of these states.

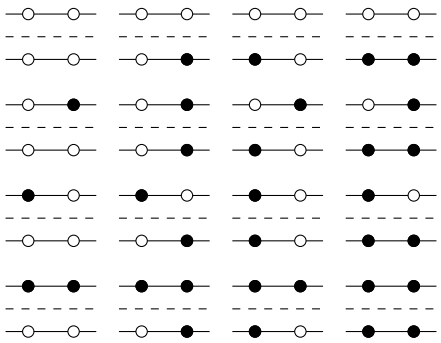

Figure 2: 16 Type-1 Basic Parallel SSEP Configurations

The full list of states of the Type-1 Basic Parallel SSEP can be seen in Figure 2. It is straightforward to show that this particle system is generated by the generator derived before, as this particle system has four absorbing states (the 4 corners), four balanced states with two choices each (the inner 2×2 square), and eight states separated into four pairs that only switch back and forth (the 8 non-corner edges), the same states in the same communicating classes that are suggested by G_2. Also, when a state has multiple choices it transitions to any one of them at an equal rate, as shown in the generator.

3.3 Expansion of the Particle System

Definition 6. *A **subsystem** of a Parallel SSEP with N sites is a subset of the system such that it is a Basic Parallel SSEP. A Parallel SSEP with N sites is made up of $N - 1$ overlapping subsystems which define the local properties of the entire system.*

Definition 7. *A **Type-m Parallel SSEP with N sites** is a Parallel SSEP with N sites such that every subsystem is a Type-m Basic Parallel SSEP.*

Lemma 11. *Let S be the set of states of a Type-m Basic Parallel SSEP such that the top-left site is fixed in state a and the bottom-left site is fixed in state b. There are then $2(m+1)$ states in S where 1 state is absorbing, 1 state is balanced, and $2m$ states that are pairwise.*

Proof. Clearly, we have 2 choices for the bottom-right site and $m+1$ choices for the top-right site giving us a total of $2(m+1)$ states. Now, note that we have one absorbing state: top-right in state a and bottom-right in state b. We also have one balanced state: top-right in state $1-a$ and bottom-right in state 0 if $b=1$ or state 1 if $b=0$. By Lemma 10 the remaining states are pairwise and so we have $2m$ pairwise states. □

Lemma 12. *A Type-m Parallel SSEP with N sites has $2(m+1)$ maximum-choice states*

Proof. Since the total number of choices of the system is the sum of the choices of its subsystems, we know that our maximum-choice states are those such that every subsystem is maximum-choice. Let our first subsystem be balanced, then by Lemma 11 we know that there is only one way for the rest of the subsystems to be balanced. Thus, the total number of maximum-choice states is simply the total number of balanced states for a Type-m Basic Parallel SSEP, which by Lemma 9 is $2(m+1)$. □

Lemma 13. *A Type-m Parallel SSEP with N sites has $2(m+1)$ absorbing states*

Proof. The proof is equivalent to the proof of Lemma 12, with absorbing states substituted for maximum-choice and balanced states. □

Theorem 2. *Let L be the generator matrix from Theorem 1, and let:*

$$L_N = \sum_{i=1}^{N-1} \underbrace{I \otimes \cdots \otimes I}_{i-1} \otimes L \otimes \underbrace{I \otimes \cdots \otimes I}_{N-1-i} \tag{1}$$

L_N is then the generator of a Type-m Parallel SSEP with N sites.

Proof. To provide motivation for the proof, note by Lemma 11 that if our first subsystem is in a given state, we then have $(2(m+1))^{N-2}$ possible states for the entire system. Since our first subsystem has $4(m+1)^2$ choices by Lemma 7 we then get $(2(m+1))^N$ total states for our system as expected. This reflects the idea that we can obtain our system's states by starting with our choices for the first subsystem, and then working our way subsystem by subsystem across the rest of our system. This idea combined with the previously stated fact that the total choices for a state of our system is the sum of the choices of the subsystems gives us our ability to prove this theorem using (1), the formula for L_N.

By Lemma 1, our generator matrix L has diagonal in the form of the multiset $\{0, \underbrace{-1, -1, \ldots, -1}_{2m}, -2m\}$ repeated $2(m+1)$ times. This can be interpreted as the $2(m+1)$ ways to choose the leftmost sites' states, where the multisets then represent the ways to finish the subsystem and their resulting choice-counts seen in Lemma 11. Now, we will show that the i^{th} term in (1) is corresponding directly to the i^{th} subsystem. We are first making $(2(m+1))^{i-1}$ diagonal copies of L to reflect the $(2(m+1))^{i-1}$ possible states of the system to the left of the i^{th} subsystem. We then take what we have so far and make $(2(m+1))^{N-1-i}$ diagonal copies of each of its diagonal elements to allow for the $(2(m+1))^{N-1-i}$ possible ways to choose the rest of the system given the i^{th} and previous subsystems have been chosen, and we note that these paths are reflected by the multiset expected from Lemma 11. Thus, by summing up the terms of (1) we are summing up the choices for each possible subsystem which completely define the entire system by Definition 7. □

Python code to build a model of this particle system is included in Appendix B, as well as a function that counts the number of choice states that emerge from the model for a given m and N value.

Acknowledgements

This research was conducted as part of the NSF-funded REU (DMS-1757872) at Texas A&M University. We would like to thank our mentor, Dr. Jeffrey Kuan, and our TA's, Ola Sobieska and Zhengye Zhou, for their work in helping us with this project.

References

[1] Gioia Carinci, Cristian Giardinà, Frank Redig, and Tomohiro Sasamoto. A generalized asymmetric exclusion process with $U_q(\mathfrak{sl}_2)$ stochastic duality. *Probab. Theory Related Fields*, 166(3-4):887–933, 2016.

[2] Jeffrey Kuan. Stochastic duality of ASEP with two particle types via symmetry of quantum groups of rank two. *J. Phys. A*, 49(11):115002, 29, 2016.

A Appendix: Generator Matrix Python Code

The following code provides functions in Python that follow all of the steps outlined in the paper to arrive at a generator matrix (G_n) of the Lie Algebra $\mathfrak{so}_{2n}$ for a given n. The LN function will take G_n and expand it to a generator matrix for N sites, using the expansion formula described in Theorem 2.

```
import numpy as np
import math
n_default = 3
# Defines matrix operations in terms of numpy operations.
def TensorProduct(A,B):
    return np.kron(A,B)
def Identity(n):
    return np.eye(n)
def MatrixProduct(A,B):
    return np.matmul(A,B)
def EmptyMatrix(n):
    return np.zeros((n,n))
# Defines H, X, Y, and Z matrices in so2n.
def H(i, n = n_default):
    m = EmptyMatrix(2*n)
    m[i-1, i-1] = 1
    m[n+i-1, n+i-1] = -1
    return m;
def X(i, j, n = n_default):
    m = EmptyMatrix(2*n)
    m[i-1, j-1] = 1
    m[n+j-1, n+i-1] = -1
    return m
def Y(i, j, n = n_default):
    m = EmptyMatrix(2*n)
    m[i-1, n+j-1] = 1
    m[j-1, n+i-1] = -1
    return m
def Z(i, j, n = n_default):
    m = EmptyMatrix(2*n)
    m[n+i-1, j-1] = 1
    m[n+j-1, i-1] = -1
    return m
# Populates a list creating the Cartan Weyl Basis for so2n given n.
def cartanweylbasis(n=n_default):
    lst = []
    for i in range(n):
        for j in range(n):
            if i == j:
                lst.append(H(i+1,n))
            if i < j:
```

```
                lst.append(X(i+1,j+1,n))
                lst.append(X(j+1,i+1,n))
                lst.append(Y(i+1,j+1,n))
                lst.append(Z(i+1,j+1,n))
    return lst
# Calculates the dual basis counterpart of a CW Basis element of so2n.
def dual(A,n=n_default):
    matrix=np.zeros((2*n,2*n))
    for i in cartanweylbasis(n):
        if (MatrixProduct(A,i)).trace() != 0:
            matrix=(1/((2*n-2)*(MatrixProduct(A,i)).trace()))*i;
    return matrix
# Calculates standard representation of Casimir (2n x 2n matrix)
def Casimir(n=n_default):
    matrix=EmptyMatrix(2*n)
    for A in cartanweylbasis(n):
        matrix = matrix + (np.matmul(A,dual(A,n)))
    return matrix
# Tensor Product Representation of an element in basis.
def representation(A,n=n_default):
    return TensorProduct(A,Identity(2*n))+TensorProduct(Identity(2*n),A)
# Calculates Casimir term that comes from one basis element. (Representation times representation of dual)
def CasimirTerm(A,n=n_default):
    return MatrixProduct(representation(A,n),representation(dual(A,n),n))
# Calculates rho(Omega), representation of Casimir element
def RhoOmega(n=n_default):
    size=4*n*n
    matrix=EmptyMatrix(size)
    for A in cartanweylbasis(n):
        matrix = matrix + CasimirTerm(A,n)
    return matrix
# Creates the C matrix, where each diagonal entry is sum of row from RhoOmega.
def C_Creator(n=n_default):
    size=4*n*n
    matrix=Identity(size)
    RO=RhoOmega(n)
    for i in range(size):
        rowtot=0
        for j in range(size):
            rowtot=rowtot+RO[i,j]
        matrix[i,i]=rowtot
    return matrix
# Subtracts of the correction term.
def RhoOmegaMinusC(n=n_default):
    matrix=RhoOmega(n)
    return matrix-C_Creator(n)
# Negates rows with positive diagonals, returning generator.
def Generator(n=n_default):
    size=4*n*n
    matrix=RhoOmegaMinusC(n)
    for i in range(size):
        if matrix[i,i]>0:
            for j in range(size):
                if matrix[i,j]!=0:
                    matrix[i,j]=-1*matrix[i,j]
    return matrix
# Expands generator to be generator matrix for N sites, takes Generator(n) and n sites.
def LN(L=Generator(),N=3):
    if (N<3 or int(N) != N):
        print("N must be an integer greater than 2")
        return
    def f(j):
        id_size=int(math.sqrt(L.shape[0]))
        if j==0:
            right_id=np.identity(id_size)
            for i in range(1,N-j-2):
                right_id=np.kron(right_id,np.identity(id_size))
            return np.kron(L,right_id)
```

```
        elif j==N-2:
            left_id=np.identity(id_size)
            for i in range(1,j):
                left_id=np.kron(left_id,np.identity(id_size))
            return np.kron(left_id,L)
        else:
            left_id=np.identity(id_size)
            for i in range(1,j):
                left_id=np.kron(left_id,np.identity(id_size))
            right_id=np.identity(id_size)
            for i in range(1,N-j-2):
                right_id=np.kron(right_id,np.identity(id_size))
            return np.kron(np.kron(left_id,L),right_id)
    return sum(f(j) for j in range(0,N-1))
```

B Appendix: Particle System Python Code

The following code builds the particle system in Python, as well as includes functions to count the number of different choice states for a given system. It uses the coordinate system (lattice, index, mass) where lattice is 0/1 for top/bottom lattice, index starts at 0 from leftmost site, and mass is an integer (take the masses from the type-m definition and scale by m). A state is then a collection of coordinates.

```
import numpy as np

def empty_state(N): #creates Parallel SSEP with N empty sites
    return np.zeros((2,N))

def auto_state(m,N,tuple_list): #auto enter particles, raises errors
    state=empty_state(N)
    for tup in tuple_list:
        lattice=tup[0]
        index=tup[1]
        mass=tup[2]
        if lattice != 0 and lattice != 1:
            raise ValueError('{} is bad (lattice)'.format(tup))
        elif index not in range(0,N):
            raise ValueError('{} is bad (index)'.format(tup))
        elif mass not in range(0,m+1):
            raise ValueError('{} is bad (mass)'.format(tup))
        elif lattice==1 and (mass != 0 and mass != m):
            raise ValueError('{} is bad (mass-lattice)'.format(tup))
        state[lattice,index]=mass
    return state

def energy_choices(state,index,shift):
    #gives the amount of choices the bottom pairing has given energy from top
    choices=1
    bottom=state[1]
    if bottom[index+shift]!=0 and bottom[index]==0:
        choices+=1
    if bottom[index]!=0 and bottom[index+shift]==0:
        choices+=1
    return choices

def create_all_lists(m,N): #gives every possible configuration given m,N
    #this function returns a list of lists of coordinate tuples
    #each list inside the list corresponds to a possible state
    #operates as a sort of decision tree to get to all possible states

    big_lst=[]
    configs=(2**N)*(m+1)**N
    for i in range(0,configs):
        big_lst.append([])
    for site in range(1,N+1):
        for mass in range(0,m+1):
            chunk=int(configs/((m+1)**site)+0.5)
            for index in range(0,chunk):
                for rotations in range(0,(m+1)**(site-1)):
                    big_lst[index+mass*chunk+rotations*(m+1)*chunk].append((0,site-1,mass))
    for site in range(1,N+1):
        for mass_scale in range(0,2):
            chunk=int(configs/(((m+1)**N)*(2**site)))
            for index in range(0,chunk):
                for rotations in range(0,((m+1)**N)*(2**(site-1))):
                    big_lst[index+mass_scale*chunk+rotations*2*chunk].append((1,site-1,mass_scale*m))
    return big_lst


def check_choices(m,state):
    #returns the number of choices a state has

    N=state.shape[1]
    top=state[0]
    bottom=state[1]
    choices=0

    #top lattice
    for j in range(0,N):
        if top[j] !=0: #if particle in jth index
```

```
            if j != 0: #if particle has a site to the left
                if (top[j]<=top[j-1]) or (top[j-1]==0): #if left site is heavier or zero
                    if top[j-1]+top[j]==bottom[j-1]+bottom[j]: #if left subsytem is in equilibrium
                        if top[j-1]==0:
                            #if left site empty, can move all over or fission for energy
                            choices+=1+(energy_choices(state,j,-1)*(top[j]-1))
                        elif top[j-1]==top[j]:
                            #if left site equal, skip to avoid double counting choices
                            pass
                        else:
                            #we can fission/fusion so need energy_choices()
                            #that min() thing is the number of ways to put a+b objects into 2 ordered boxes with max capacity m (a,b < m)
                            choices+=(energy_choices(state,j,-1)*min(top[j]+top[j-1],2*m-top[j]-top[j-1]))
                    elif top[j]!=top[j-1]:
                        #if not in equilibrium and left site is heavier/zero, 1 choice
                        choices+=1
            if j != N-1: #if a particle has a site to the right
                if (top[j]<=top[j+1]) or (top[j+1]==0): #if right site is heavier, equal or zero
                    if top[j+1]+top[j]==bottom[j+1]+bottom[j]: #if right subsytem is in equilibrium
                        if top[j+1]==0:
                            #if right site empty, can move all over or fission for energy
                            choices+=1+(energy_choices(state,j,1)*(top[j]-1))
                        elif top[j+1]==top[j]:
                            #if right site equal
                            #same idea as else: but with a=b
                            choices+=(energy_choices(state,j,1)*min(2*top[j],2*m-2*top[j]))
                        else:
                            #same as previous else but for right
                            #isn't double counting since a<b or b<a (will only do this once for pairing)
                            choices+=(energy_choices(state,j,1)*min(top[j]+top[j+1],2*m-top[j]-top[j+1]))
                    elif top[j]!=top[j+1]:
                        #if not in equilibrium and right site is heavier/zero, 1 choice
                        choices+=1
    #bottom lattice
    for j in range(0,N):
        if bottom[j] !=0: #if particle occupies site
            if j!=0 and (((top[j]==m or top[j]==0) and (top[j-1]==m or top[j-1]==0)) or ((top[j-1]==top[j]) and 2*top[j]!=bottom[j])) and bottom[j-1]==0:
                #if left site exists, and is open, and the bottom particle is allowed to move
                choices+=1
            if j!=N-1 and (((top[j]==m or top[j]==0) and (top[j+1]==m or top[j+1]==0)) or ((top[j+1]==top[j]) and 2*top[j]!=bottom[j])) and bottom[j+1]==0:
                #if right site exists, and is open, and the bottom particle is allowed to move
                choices+=1

    return int(choices+0.5) #int to avoid float problems

def count_states(m,N):
    #tallies up number of j-choice states as dictionary similar to so_2n.py
    configs=create_all_lists(m,N)
    states={}
    for config in configs:
        choices=check_choices(m,auto_state(m,N,config))
        if choices in states:
            states[choices]+=1
        else:
            states[choices]=1
    return states

def print_states(states):
    #prints count_states result in order
    ordered_choices=sorted(states.keys())
    for choice in ordered_choices:
        print("{}:{}".format(choice,states[choice]))
```