**Symmetric functions**

**Harper Niergarth, Vixail Hadelyn, Wenhui Li, Wieyou Li**

Being written by
**Vixail Hadelyn**

Faculty of Mathematics
University of Waterloo
Canada, Ontario
October 10, 2025

# Contents



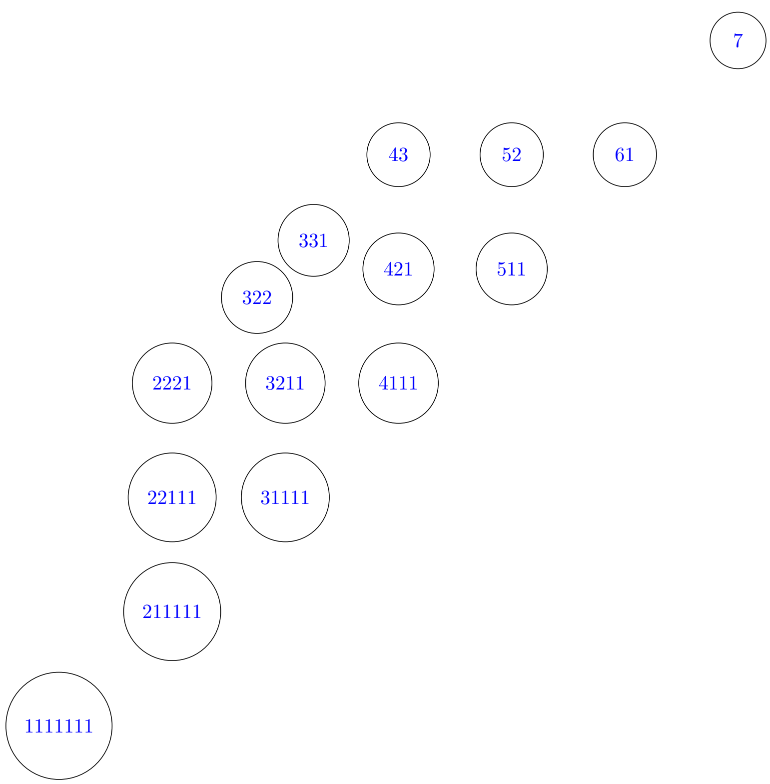
# 1 Introduction

# 2 Known results

## 2.1 Partitions

Recall that a partition is a collection of parts of length $\ell$ with sum of the parts being the **size**.
In essence a partition $\lambda \in \mathbb{Z}^\ell / S_\ell$, as any permutation of a partition is the same.
As a unique representative from $\mathbb{Z}^\ell$, $\lambda = (\lambda_1, ..., \lambda_\ell)$ is chosen to be a weakly decreasing sequence.
This is unique, as the only permutations which keep the sequence decreasing are those which "swap" adjacent equal parts.

To find all the partitions with fixed size, there are multiple methods to do this, although it is difficult to ensure no repetition making counting partitions with fixed size a difficult problem.
As such there are many different ways to plot and connect partition. Below is one where the column and row is determined by the largest part, and the length. This highlights the two main properties under consideration, show the symmetry of a conjugate partition, and that some partitions like 311 and 322 are not uniquely determined by their length and largest part.
In a sense the location of each circle is the bounding box of a young-diagram.



**Code** [4]**.** In the source code [4], a `Partition(multiset[int])` is a class which implements the desired abstract behaviour defined above, and inherits behaviour from the base multiset class `Counter from collections`.
A partition is initialized the same way a `Counter` is, by being passed a list of elements.

A partition has the following additional properties

- `Partition(...).parts`: The composition given by ordering the partition
- `Partition(...).isempty`: If the partition is empty
- `Partitions(...).sum`: Returns the total of all the elements
- `Partitions(...).isType(type: CTYPE, j: int)`: Determines if the partition is an
  - Elementary: Is an indicator, 0 or 1, given by a subset of size j
  - Monomial: Is a general weighting with total weight j
  - Polynomial: Gives one term a value of j and all others 0

## 2.2 Composition

**Code** [4]**.** 4.1

## 2.3 Symmetric polynomials

A quick way to see that $\mathtt{Sym}_n$ is closed under addition is to see that by taking permutations on some shared input to $f, g \in \mathtt{Sym}_n$, the output is the same.
This means $f + g$ and $f * g$ are both fixed.

Recall the definition for the $j$th symmetric polynomial

$$e_j(x_1, ..., x_n) = \sum_{1 \leq i_1 < ... < i_j \leq n} x_{i_1} ... x_{i_j}$$

Most importantly $e_j \in \mathtt{Sym}_n$ with degree $j$.

**Theorem 2.1.** *[The fundamental theorem of symmetric polynomials] Every $f \in \mathtt{Sym}_n$ may be written uniquely as $\mathbb{Q}$-linear combination of finite products of $\{e_1, e_2, ..., e_n\}$.*

*Proof.* Consider $f = \sum_\lambda c_\lambda x[\lambda]$ with $x[\lambda] = \sum_{\sigma \in S_n} x_{\sigma(1)}^{\lambda_1} ... x_{\sigma(n)}^{\lambda_n}$.
Now consider the terms of largest degree $k = \deg(f)$, meaning $|\lambda| = k$. Consider $\lambda$ with non-zero $c_\lambda$ and largest parts.
By taking $f - c_\lambda \prod_{i=1}^k e_{\lambda_i'}$, then the $c_\lambda$ term will go to 0.
This is because $\prod_{i=1}^k e_{\lambda_i'}$ This also won't effect any terms above, and so by repeating this process the terms in $f$ of degree $k$ go to 0.
Then by repeating this inductively on the degree $k$, $f$ becomes constant ∎

## 2.4 Injectivity

### 2.4.1 d-ary partitions

**Conjecture 2.1** (Cimpoease and Tanase [1]). If $\lambda, \mu$ are $d$-ary partitions with $l$ parts s.t. $1 \leq j \leq l-1$ and $\texttt{pre}_j(\lambda) = \texttt{pre}_j(\mu)$, then $\lambda = \mu$

In fact this case is disproven by a counterexample found by Wen.
Notice that $\texttt{pre}_3(8, 8, 2, 2, 2, 1) = \texttt{pre}_3(8, 4, 4, 4, 1, 1)$

### 2.4.2 The $j = 2$ case

**Theorem 2.2.** *If $\lambda, \mu$ are partitions of $n$ s.t. $\texttt{pre}_2(\lambda) = \texttt{pre}_2(\mu)$, then $\lambda = \mu$*

*Proof.* This proof follows from the work shown by Jiahui Li [2].

□

# 3 The Homogeneous Analog

From the unpublished preliminary reading [3], a few exercises and definitions were considered. Importantly

**Definition 3.1.** The pre partition function with repeats is given as

$$\mathtt{prh}_j(\lambda) = \{\{\lambda_{i_1}...\lambda_{i_j} : 1 \leq i_1 \leq ... \leq i_j \leq n\}\}$$

**Corollary 3.0.1.** $\mathfrak{l}(\mathtt{prh}_j(\lambda)) = \binom{n+j-1}{j}$

## 3.1 Injectivity

**Proposition 3.1.** $\mathtt{prh}_j$ is injective for $j > 0$, meaning if $\mathtt{prh}_j(\lambda) = \mathtt{prh}_j(\mu)$, then $\lambda = \mu$

*Proof.* Let $\lambda$ and $\mu$ be two arbitrary partitions such that $\mathtt{prh}_j(\lambda) = \mathtt{prh}_j(\mu)$.
Then by induction on $i$, it will be shown that $\lambda_i = \mu_i$

**Base Case**

For the base case $\lambda_1 = (\max(\mathtt{prh}_j(\lambda)))^{\frac{1}{j}} = \mu_1$, which requires $j > 0$

**Inductive step**

Recall the inductive hypothesis that for $i < n : \lambda_i = \mu_i$, and consider the following notation.
Let the elementary degree $j$ monomials be given by

$$e_{(x)}(\lambda) = e_{(x_1,...,x_j)}(\lambda) = \lambda_{x_1}\lambda_{x_2}...\lambda_{x_j} \text{ where } x_1 \leq x_2 \leq ... \leq x_j$$

Notice that the largest term of $\mathtt{prh}_j(\lambda)$ with at least one unknown term is $\lambda_1^{j-1}\lambda_n$. This means

$$\lambda_1^{j-1}\lambda_n = \max(\{e_{(x)}(\lambda)\}\backslash\{e_{(x)}(\lambda) : x_j < n\}) = \max(\{e_{(x)}(\mu)\}\backslash\{e_{(x)}(\mu) : x_j < n\}) = \mu_1^{j-1}\mu_n$$

Importantly by the inductive hypothesis $\{e_{(x)}(\lambda) : x_j < n\} = \{e_{(x)}(\mu) : x_j < n\}$
In addition $\{e_{(x)}(\lambda)\} = \mathtt{prh}_j(\lambda) = \mathtt{prh}_j(\mu) = \{e_{(x)}(\mu)\}$

Overall, this means that by induction $\lambda = \mu$, and $\mathtt{prh}_j$ is injective for $j > 0$.
In addition, since this proof contains functions to determine each $\lambda_i$, then this serves as a method for an algorithm to compute $\lambda$ given $\mathtt{prh}_j(\lambda)$ and $j$. □

## 3.2 Surjectivity

It may not be too surprising that $\mathtt{prh}_j$ is not surjective as it "stretches" inputs often making them larger.
Notably, the output length must be $\binom{\mathfrak{l}(\lambda)+j-1}{j}$, and the largest term must be $\lambda_1^j$.
Other than this one can look at exactly where the algorithm fails, to see why a certain partition is not an output.
The following are some examples

| | |
|---|---|
| $j = 2, \mu = (2)$ | Reason: $2^{\frac{1}{2}}$ is not an integer |
| $j = 2, \mu = (1, 1, 1, 1, 1)$ | Reason: Improper length |
| $j = 3, (27, 18, 8)$ | Reason: Missing $12 = 3 * 2 * 2$ |
| $j = 3, (27, 12, 4, 2)$ | Reason: $\dfrac{12}{9}$ is not an integer |

Interestingly, so of these do exists when the underlying number system is changed. The only problems that wouldn't fix are missing numbers, or improper length.
If the rationals are used then $j = 3, \mu = (27, 12, 4, \frac{64}{27})$ has inverse $(3, \frac{4}{3})$.
A very nice property is that by multiplying the output by some $x^3$, then the input scales by $x$, and this problem above is functionally the same as $\texttt{prh}_3((81, 4)) = (729, 324, 108, 64)$.

## 3.3 New perspectives and definitions

**Definition 3.2.** A **combination** $(\alpha_1, \alpha_2, ..., \alpha_s)$ is a possibly zero integer-sequence.
A strict combination is one with no zeroes.
Importantly a combination can be taken to a permutation by **sorting** which gives decreasing $(\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_n})$ which in these notes removes zero.
Take the set of all compositions of length $s$ be $\texttt{Comp}(s)$

**Definition 3.3.** Let the monomial given by a composition $\alpha$ be

$$x^\alpha = (x_1, x_2, ..., x_n)^{(\alpha_1, \alpha_2, ..., \alpha_n)} = x_1^{\alpha_1} x_2^{\alpha_2} * ... * x_n^{\alpha_n}$$

**Remark.** $x^\alpha$ Acts normally under distribution and exponentiation

**Definition 3.4.** $m_\lambda(x_1, ..., x_n) = \sum_{\alpha \in \texttt{Comp}(n) : \texttt{sort}(\alpha) = \lambda} e_\alpha$

Importantly, 2.1 uses the fact that $m_\lambda$ must be a linear basis for $\texttt{Sym}_n$.
And the terms of $m_\lambda$ which are given by $e_\alpha = x_1^{\alpha_1} x_2^{\alpha_2} * ... * x_n^{\alpha_n}$ form the standard basis for $\mathbb{Z}[x_1, x_2, ..., x_n]$.

For the symmetric functions we study, $e_\alpha$, forms the basis, and in fact exact terms of the output partition up to given coefficients.
This is done by taking $e_\alpha(\lambda_1, ..., \lambda_n) = e_\alpha(\lambda)$.
In addition, if some $e_\alpha$ is present, then $e_{\sigma(\alpha)}$ is present as well since $\texttt{sort}(\sigma(\alpha)) = \texttt{sort}(\alpha)$.

Now consider some arbitrary symmetric function $f(x_1, ..., x_n)$, which in the $m_\mu$ basis has coefficients $c_\mu \in \{0, 1\}$, and define

**Definition 3.5** (Generic pre symmetric function)**.** For a symmetric function $f$

$$\texttt{prf}(\lambda) = \{\{c_\alpha \lambda^\alpha : \alpha \in \texttt{Comp}(n), c_\alpha \neq 0\}\} \text{ where } c_\alpha = [x^\alpha](f)$$

**Code** [4]**.**

# 4 Ordering

To further interrogate $\texttt{prf}(\lambda)$, further structure must be put on the elements $\lambda^{\alpha}$, or more precisely $\alpha$.

**Definition 4.1** (Partial ordering of compositions)**.** It is said that $\alpha > \beta$ iff
$\hat{\alpha} \neq \hat{\beta}$ and $\forall \lambda : \lambda^{\hat{\alpha}} \geq \lambda^{\hat{\beta}}$ where $\hat{\alpha} = \frac{\alpha}{\texttt{tot}(\alpha)}$
and the length of $\lambda$ must have the same length as $\alpha$ and $\beta$.

**Code** [4]**.** symm.py overloads the definition of == and > for the `Composition` class.

```
#Examples
Composition([1,2,3]) == Composition([2,4,6]); True
Composition([0,0,0]) == Composition([2,4,6]); False
Composition([1,2,3]) == Composition([2,4,5]); False
Composition([3,2,0]) > Composition([2,2,1]); True
Composition([1,0,0]) > Composition([4,0,1]); True
Composition([2,0,1]) > Composition([1,2,0]); False
Composition([1,2,0]) > Composition([2,0,1]); False

```

**Remark.** Notice that not all compositions can be compared
as an example consider $\alpha = (2,0,1)$ and $\beta = (1,2,0)$.
For $\lambda = (2,2,1)$: $\lambda^{(2,0,1)} < \lambda^{(1,2,0)}$ as $2^2 * 1^1 \leq 2^1 * 2^2$
And $\lambda = (5,1,1)$: $\lambda^{(1,2,0)} < \lambda^{(2,0,1)}$ as $5^1 * 1^2 < 5^2 * 1^1$.
This means $(1,2,0) \not> (2,0,1)$ and $(2,0,1) \not> (1,2,0)$

**Definition 4.2.** We say that $\mathfrak{O}(\Omega)$ is the ordering graph of $\Omega$, and call $\Omega$ the system of compositions.
$V(\mathfrak{O}(\Omega)) = \Omega$ and $E(\mathfrak{O}(\Omega)) = \{\overrightarrow{\alpha\beta} : \alpha, \beta \in \Omega, \alpha > \beta\}$.

**Definition 4.3.** In addition, the implicit ordering graph $\mathfrak{O}_{\mathfrak{I}}(\Omega)$ is a spanning subgraph where:
$\overrightarrow{\alpha\beta} \in E(\mathfrak{O}(\Omega))$ if and only if there is an $\alpha$-$\beta$ path in $\mathfrak{O}_{\mathfrak{I}}(\Omega)$.
$\mathfrak{O}_{\mathfrak{I}}(\Omega)$ is often taken to be edge minimal if such a graph exists.

**Code** [4]**.** graph.py implements the `CompositionGraph` class defines an object which stores the graph structure of $\mathfrak{O}(\Omega)$ and similar directed graphs with compositions as their vertices.
`CompositionGraph` implements the following behaviours

- `CompositionGraph(...).detail`: Defines if the ordering is implicit, $\mathfrak{O}_{\mathfrak{I}}(\Omega)$, or complete, $\mathfrak{O}_{\mathfrak{I}}(\Omega)$.
- `CompositionGraph(...).tikz_str`: Returns a LaTeX tikzpicture environment which displays the graph using constant parameters `CompositionGraph._tikz_settings`
- `CompositionGraph(...).below(key)`: Gives back the set of compositions in the graph which are below the composition specified by the key

In addition, the `symm_graph` class defines functions and algorithms to construct and process `CompositionGraph` objects.

`symm_graph` implements

- `symm_graph.connect(nodes, detail: optional)`: Generates either $\mathfrak{O}(\Omega)$ or $\mathfrak{O}_{\mathfrak{J}}(\Omega)$ where `nodes` are $\Omega$, and `detail` describes whether the graph is implicit or not.

## 4.1 Examples

Consider the following subgraph of $\mathfrak{O}_{\mathfrak{J}}(\Omega)$ induced by system of compositions being all compositions of length 2 with less than 5 as their maximum

**Code [4].** The code to generate the following tikz string is

```python
nodes = list(Composition.generate(5, 2))
symm_graph.normalize(nodes)
graph = symm_graph.connect(nodes, PO_DETAIL.Implicit)
print(graph.tikz_str)

```

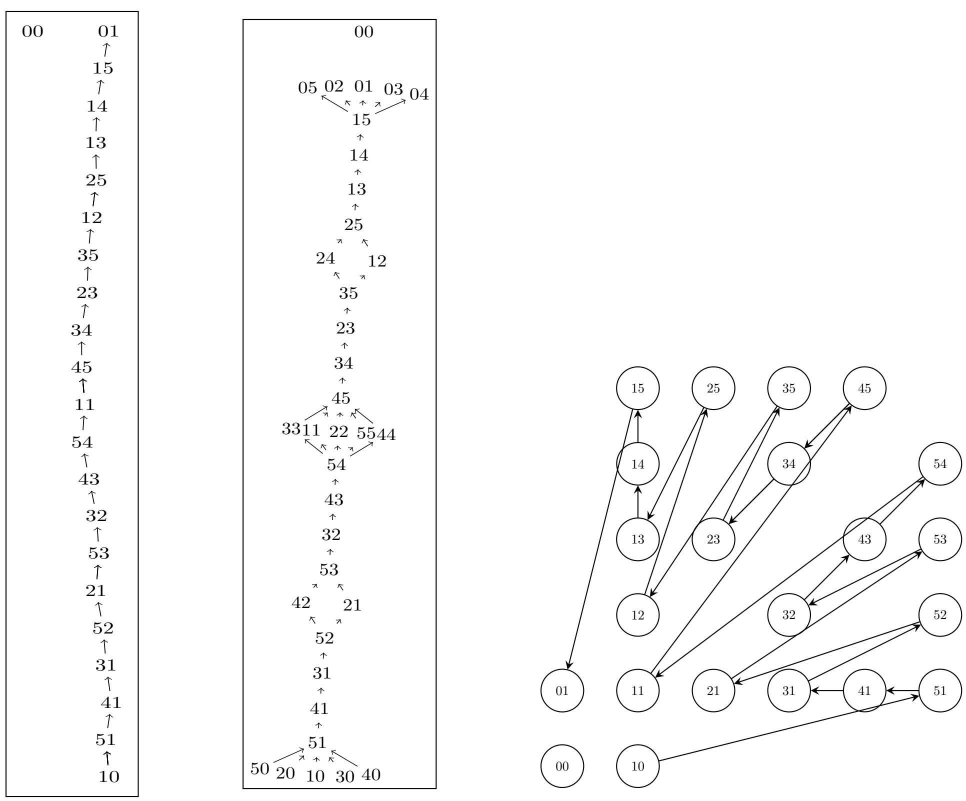



Figure 1: Automatic placing on the left, the same graph but without the normalization step in the middle, and manual placement on the right

Notice how this graph is a simply linear ordering which is missing (0,0) as it cannot be normalized and so cannot be compared.
If the graph wasn't normalized it would look like

Now for another example, examine $\mathfrak{O}_I$ for length 3 compositions and a maximum sum of 5.

**Code** [4]**.** Again here is the code to generate the tikz string for this graph

```
nodes = list(filter(lambda x: x.sum <= 5, Composition.generate(5, 3)))
symm_graph.normalize(nodes)
graph = symm_graph.connect(nodes, PO_DETAIL.Implicit)
print(graph.tikz_str)

```

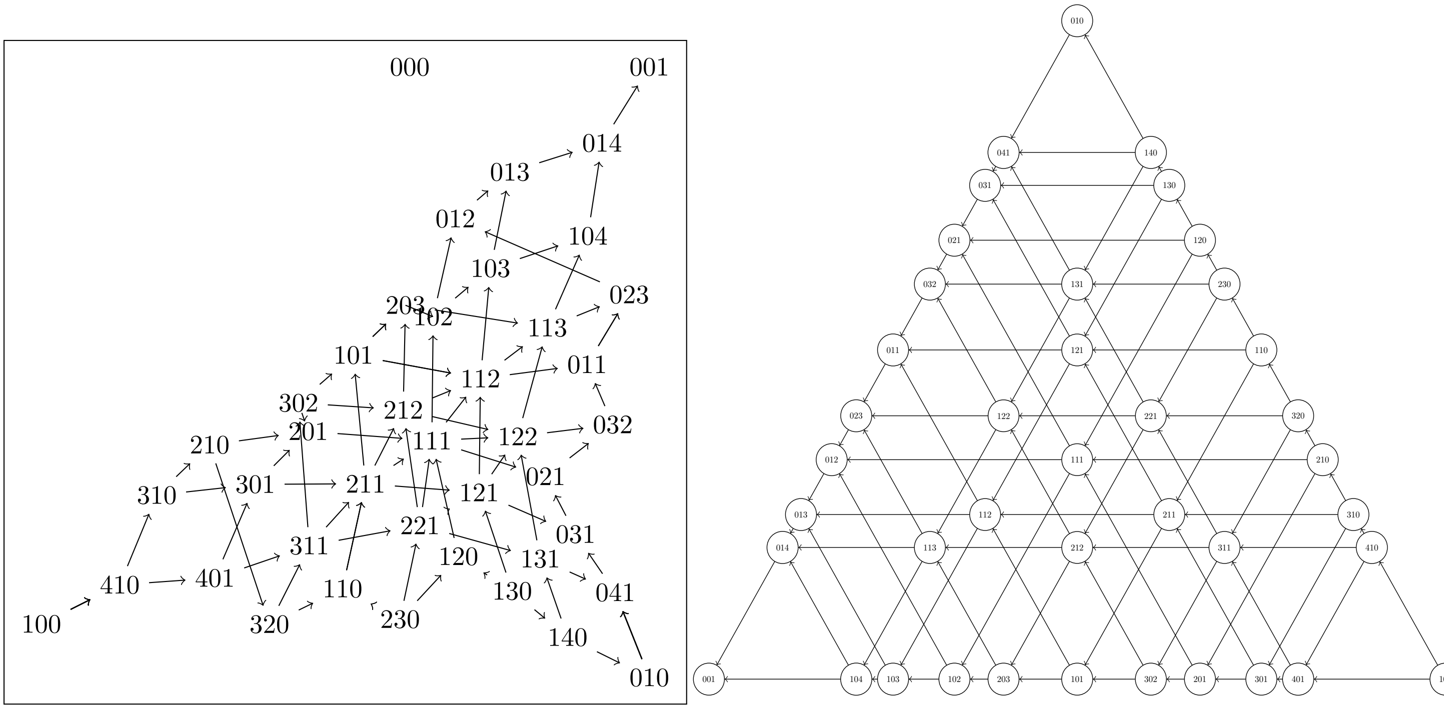



Figure 2: Again the auto generated graph is drawn on the left, with a custom graph on the right

It is clear that the graph on the right is much cleaner, although it has uneseccary edges, and took at least an hour to construct, where the graph on the left was generated in seconds.

## 4.2 Reconstructing a total ordering

The goal of introducing $\mathfrak{O}(\Omega)$ is when given values for $\lambda^\omega, \omega \in \Omega$ to either reconstruct a unique $\lambda$, or show that multiple exist.

**Remark.** $\lambda$ induces a total order on $\mathfrak{O}(\Omega)$. This is done taking $\lambda^\alpha \geq \lambda^\beta$.
In essence the partial ordering was constructed from incomplete information about $\lambda$, and so when $\lambda$ is known the partial ordering becomes a total ordering.
Furthermore, if every $\lambda^\omega$ is known, then $\lambda^\alpha$ is known for $\alpha \in \mathtt{span}_{\mathbb{R}}(\Omega)$.

**Conjecture 4.1.** Every valid total order on $\mathfrak{O}(\Omega)$ reconstructs no more than one $\lambda$.

**Conjecture 4.2.** By inductively placing elements of $\lambda^{\Omega}$ into a total order on $\mathfrak{O}(\Omega)$ and recomputing the partial order on $\mathfrak{O}(\Omega)$ at each step, a valid total order is found.

**Conjecture 4.3.** A total order on $\mathfrak{O}(\Omega)$ for a spanning $\Omega$, gives a radical rational expression for $\lambda$ in terms of $\texttt{sort}(\lambda^{\Omega})$.

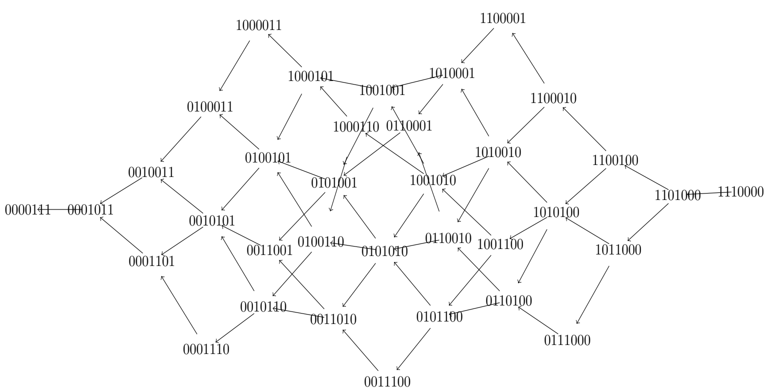
**Corollary 4.0.1.** If $\Omega$ is a system of rational compositions, then for any such radical expression above, some $\Lambda = \lambda^{\Omega}$ exists s.t. $\lambda$ is integer

### 4.3 The $\texttt{pre}_3(\lambda)$ length 7 case

To begin recall that $\texttt{pre}_3(\lambda)$ where $\lambda$ is length 7 has terms determined by composition who;s parts are an elementary partition $(1, 1, 1)$.

**Code [4].** The following generate $\Omega$ as `nodes` and prints the tikz string to display the graph below.

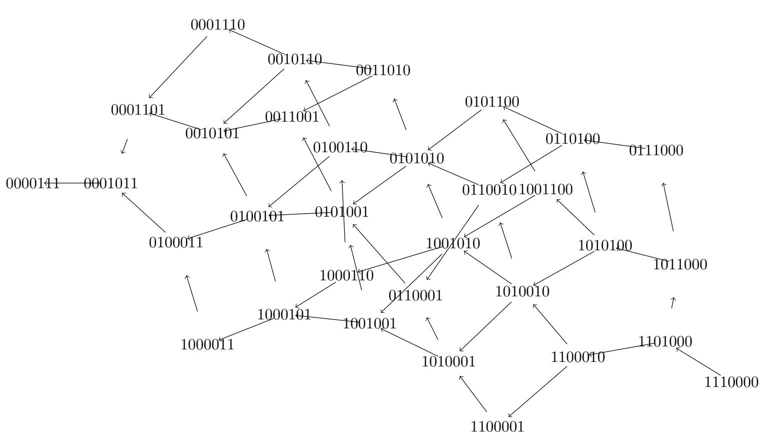
```
nodes = list(filter(lambda y:  y.order.isType(CTYPE.Elementary, 3),
 Composition.generate(3, 7)))
graph = symm_graph.connect(nodes, PO_DETAIL.Implicit)
print(graph.tikz_str)

```




To begin, notice that any total ordering on $\mathfrak{O}_{\mathfrak{J}}$ must have $1110000, 1101000$ and $0001011, 0000111$ last. Then either 1100100 or 1011000 come next in the total ordering.
Notice that $1100100 = 1101000 + 0000111 - 0001011$, and so it's exact value is already fixed by the ordering. In addition, $0010011 = 0001011 + 1110000 - 1101000$

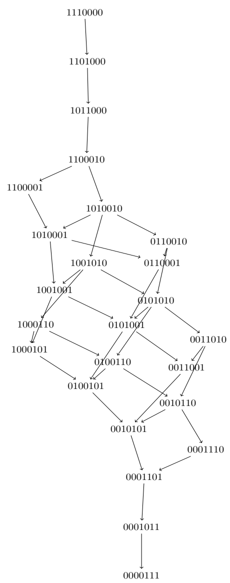
and $0011100 = 3 * \sum_{\omega \in \Omega} \omega - 1101000 - 0001011 - 2 * -1110000 - 2 * 0000111$ This means that these values can removed from $\Omega$ and the procedure can continue.

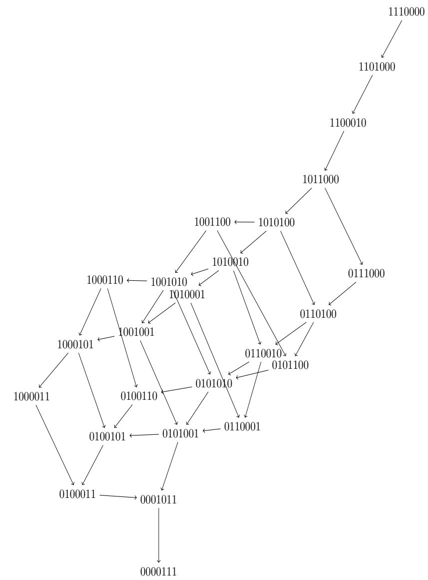
Now notice that there are exactly two options for which is third.



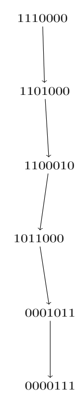

Either 1011000 or 1100010.

Since both are free and neither are restricted to come before the other by the partial ordering, they give two possible additions to the total ordering.
These look like:




Then since there's only one choice for the fourth value, this means that graph can be reduced
Finally becoming


Notice that these final derived values form a basis form $\mathbb{R}^7$ and so $\lambda$ is known from these fixed values

$$\begin{matrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{matrix} \qquad \begin{matrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{matrix}$$

# 5 Code

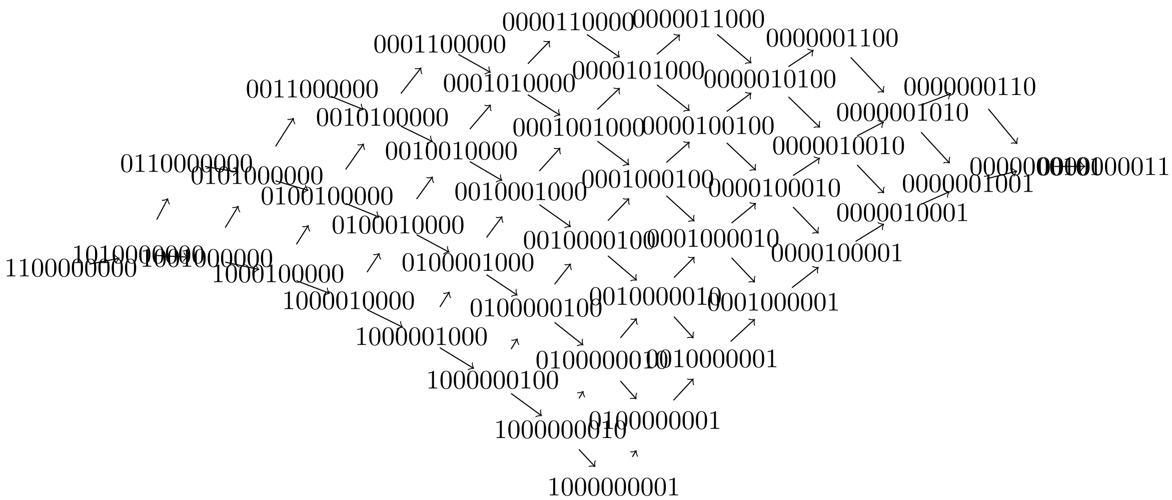
The goal of this section is too describe the methods used to implement the above procedure in code. This has not yet been implemented but is a work in progress. The source code can be found at [4]

# 6 Formalism

These examples above show an idea for computing algebraic expressions for $\lambda$ given case analysis. This section hopes to provide a rigorous method to interrogate specific problems, and relationships between solutions.

To begin ...

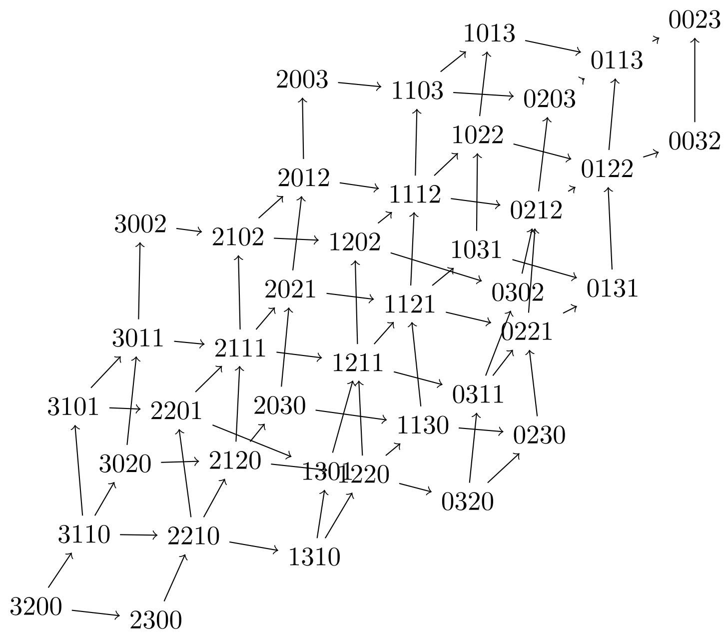
# 7 Additional graphs and diagrams




**Code [4].** Python code:

```
    nodes = list(filter(lambda y:  y.order.isType(CTYPE.Elementary, 2),
         Composition.generate(2, 10)))
    graph = symm_graph.connect(nodes, PO_DETAIL.Implicit)
    print(graph.tikz_str)
```

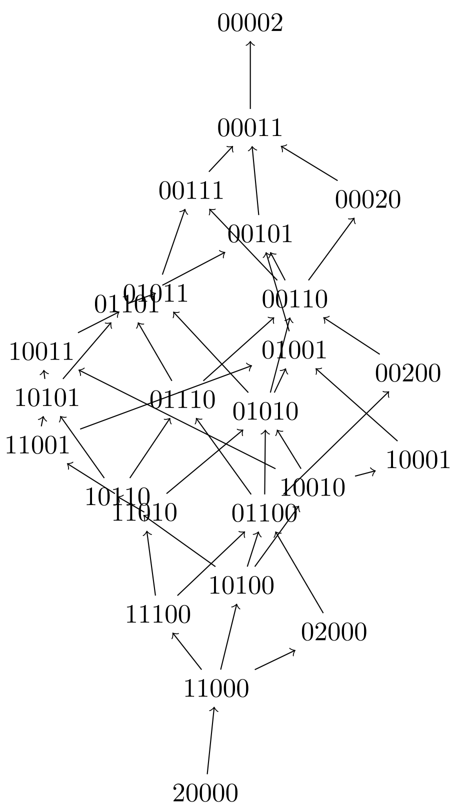
Figure 3: Elementary partitions of length 10 and choice 2




**Code** [4]**.** Python code:

```
nodes = list(filter(lambda y:  y.order.isType(CTYPE.Monomial, 5),
     Composition.generate(3, 4)))
graph = symm_graph.connect(nodes, PO_DETAIL.Implicit)
print(graph.tikz_str)

```

Figure 4: Monomial partitions of size 5, length 4, and maximum exponent 3




**Code** [4]**.** Python code:

```
nodes = list(filter(lambda y:  y.order.isType(CTYPE.Monomial, 2)
        or y.order.isType(CTYPE.Elementary, 3),
    Composition.generate(5, 5)))
graph = symm_graph.connect(nodes, PO_DETAIL.Implicit)
print(graph.tikz_str)
```

Figure 5: Monomial compositions of size 2 and length 7, and Elementary Compositions of choice 3 and length 7


**Code [4].** Python code:

```python
nodes = list(filter(lambda y:  y.order.isType(CTYPE.Elementary, 4),
     Composition.generate(4, 10)))
graph = symm_graph.connect(nodes, PO_DETAIL.Implicit)
print(graph.tikz_str)
```

Figure 6: Elementary compositions of size 2 and length 7

## References

[1] M. Cimpoeas and R. Tanase, "Remarks on $d$-ary partitions and an application to elementary symmetric partitions," 2025. [Online]. Available: https://arxiv.org/abs/2506.04459

[2] S. J. Li, "A note on multiset reconstruction from sum and pairwise products," 2025. [Online]. Available: https://arxiv.org/abs/2508.00971

[3] H. Niergarth, "Fall 2025 wim drp: Plugging partitions into symmetric functions," 2025.

[4] V. Hadelyn, "A method to determine the injectivity of symmetric functions," 2025. [Online]. Available: https://github.com/lynhadelyn/Symmetric-functions/tree/main/lib