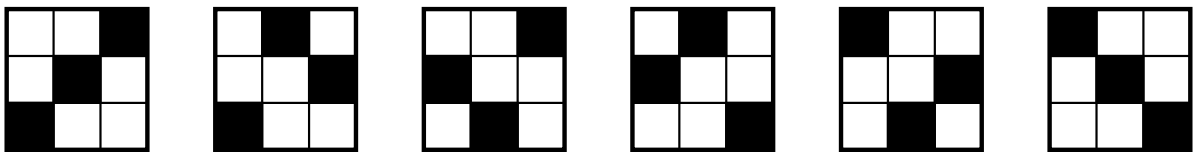
Sorting with C-machines

Jay Pantone

Dartmouth College

Hanover, NH



Erwin Schrödinger Institute
Algorithmic and Enumerative Combinatorics

October 20, 2017

Joint work with Michael Albert, Cheyne Homberger, Nathanial Shar, Vince Vatter

Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

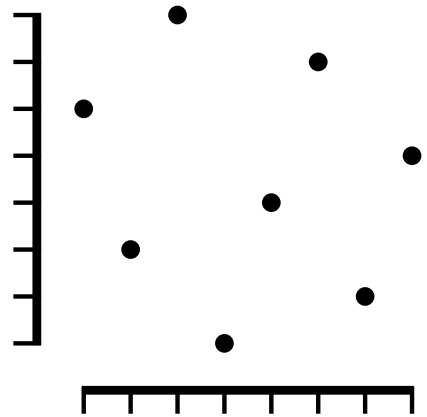
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



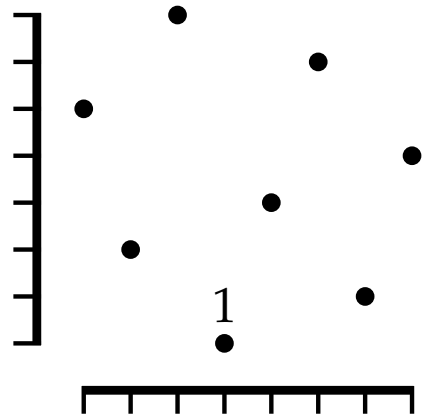
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



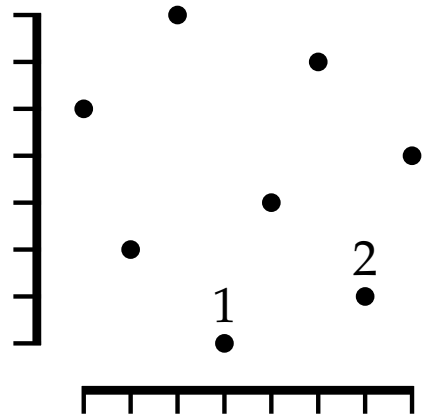
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



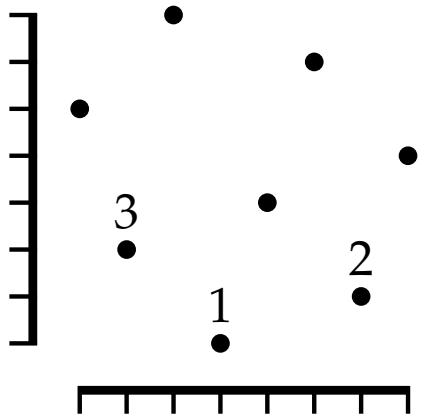
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



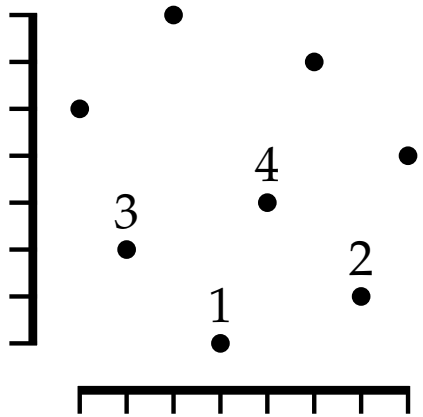
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



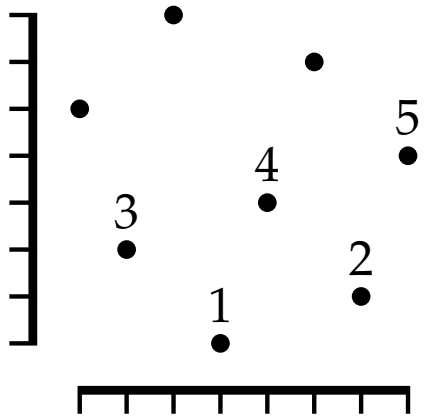
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



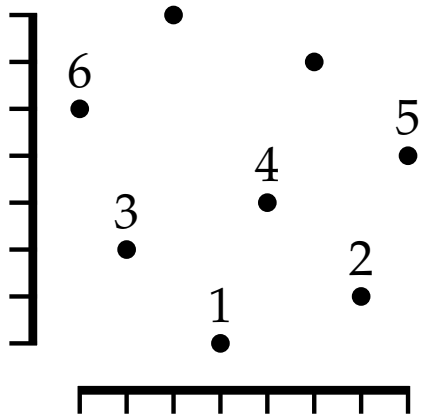
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



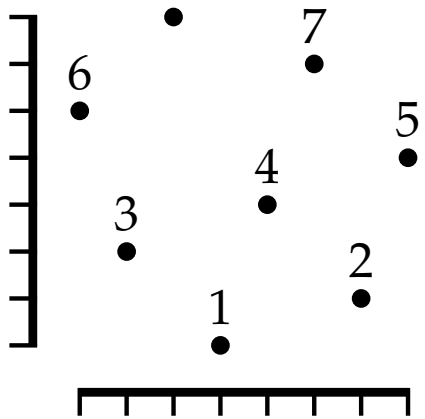
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



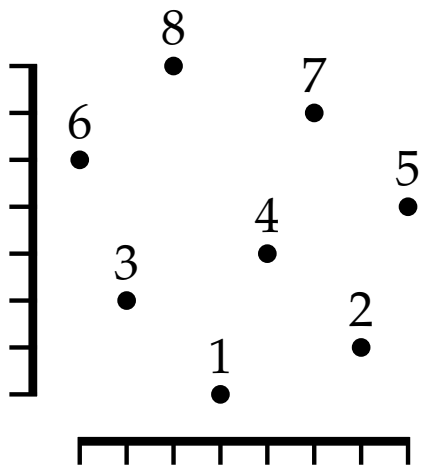
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



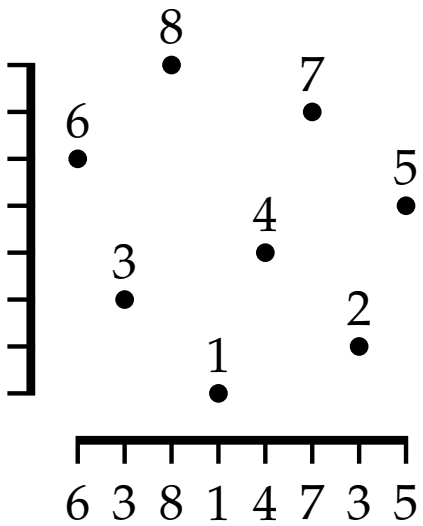
Permutations

A *permutation* of length n is viewed as an ordering of the integers $\{1, \ldots, n\}$.

- *Example:* 63814725 is a permutation of length eight.

Geometric interpretation: A permutation can be viewed as points in the plane, no two of which lie on the same horizontal or vertical line.

- *Example:* 63814725 $\Longleftrightarrow$



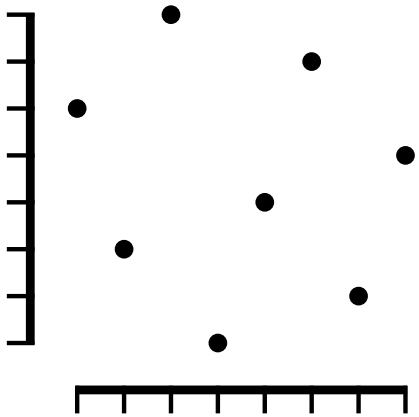
Pattern Containment

A permutation π contains a permutation σ (denoted $\sigma \leqslant \pi$) if you can delete dots in the picture of π and shrink the picture to get the picture of σ.

Pattern Containment

A permutation π contains a permutation σ (denoted $\sigma \leqslant \pi$) if you can delete dots in the picture of π and shrink the picture to get the picture of σ.

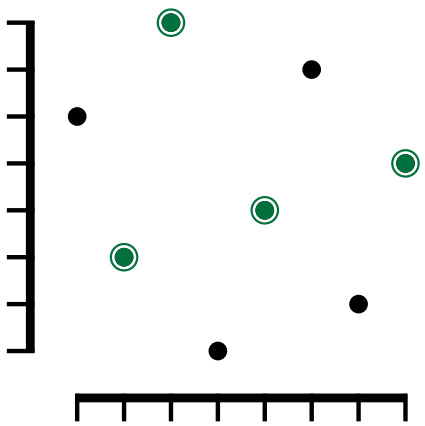
$1423 \leqslant 63814725 \iff$



Pattern Containment

A permutation π contains a permutation σ (denoted $\sigma \leqslant \pi$) if you can delete dots in the picture of π and shrink the picture to get the picture of σ.

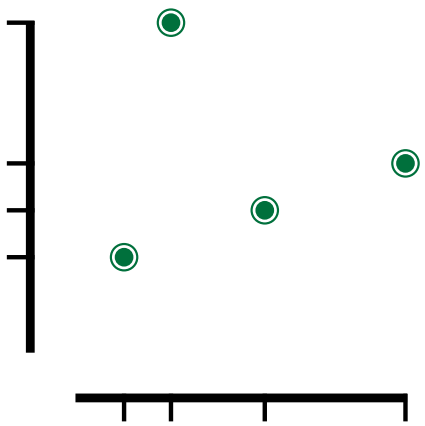
$1423 \leqslant 63814725 \iff$



Pattern Containment

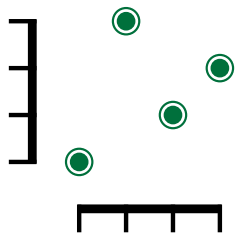
A permutation π contains a permutation σ (denoted $\sigma \leqslant \pi$) if you can delete dots in the picture of π and shrink the picture to get the picture of σ.

$1423 \leqslant 63814725 \iff$



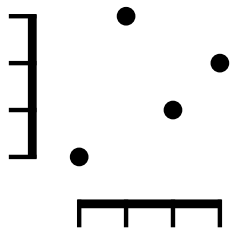
Pattern Containment

A permutation π contains a permutation σ (denoted $\sigma \leqslant \pi$) if you can delete dots in the picture of π and shrink the picture to get the picture of σ.

$1423 \leqslant 63814725 \iff$ 

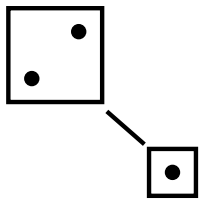
Pattern Containment

A permutation π contains a permutation σ (denoted $\sigma \leqslant \pi$) if you can delete dots in the picture of π and shrink the picture to get the picture of σ.

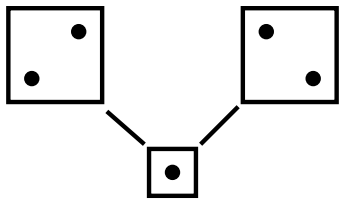
$1423 \leqslant 63814725 \iff$ 

Permutation Poset

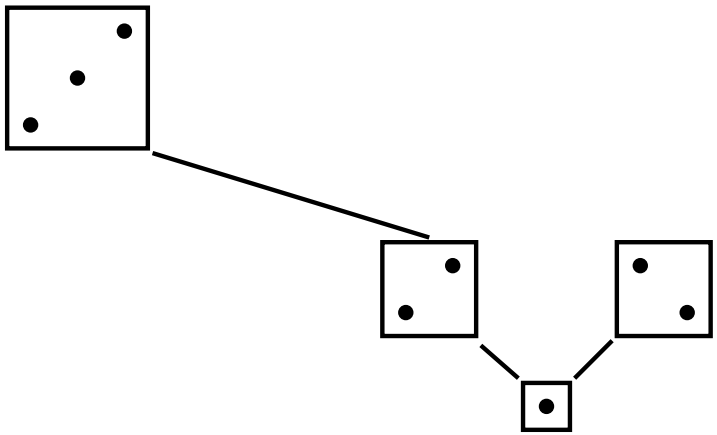
Permutation Poset



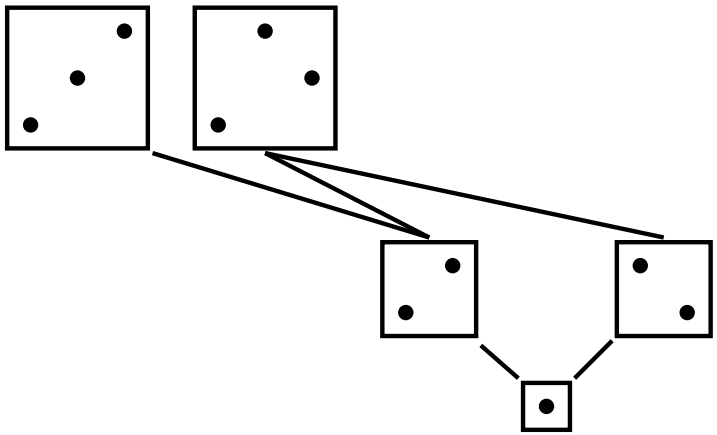
Permutation Poset



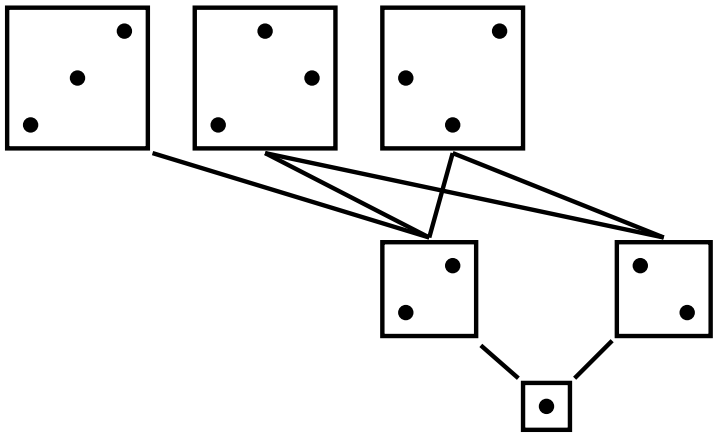
Permutation Poset



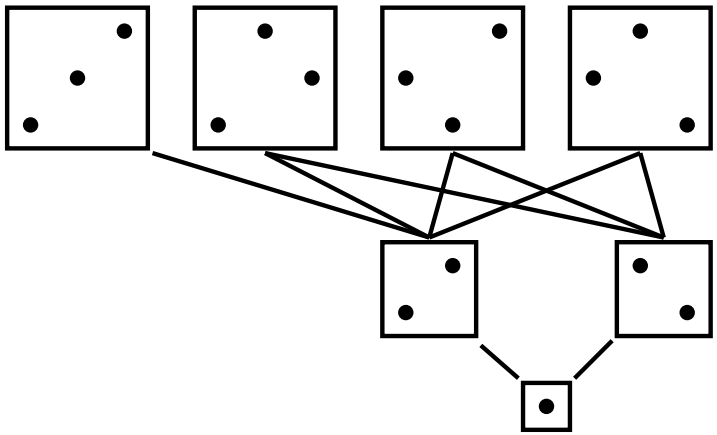
Permutation Poset



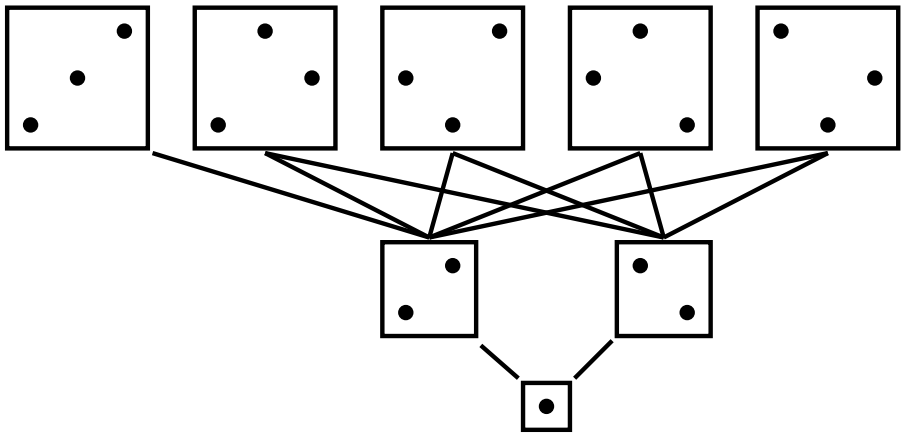
Permutation Poset



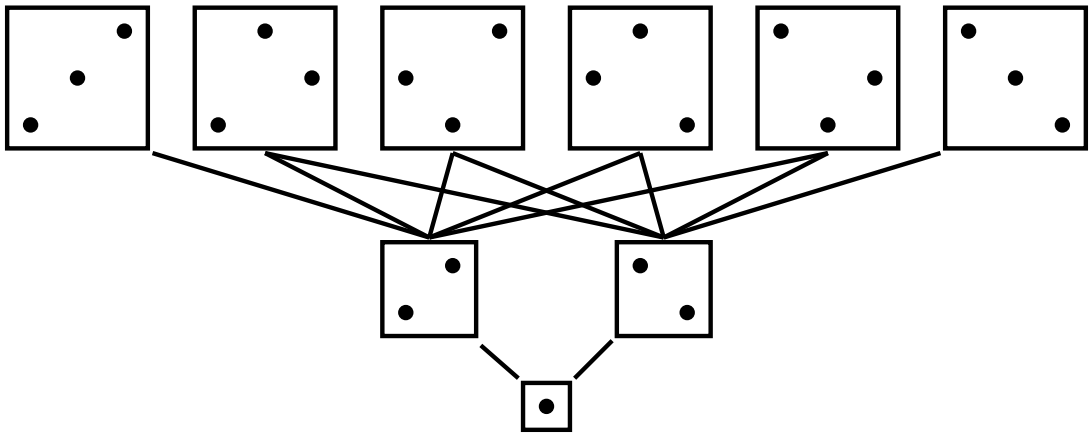
Permutation Poset



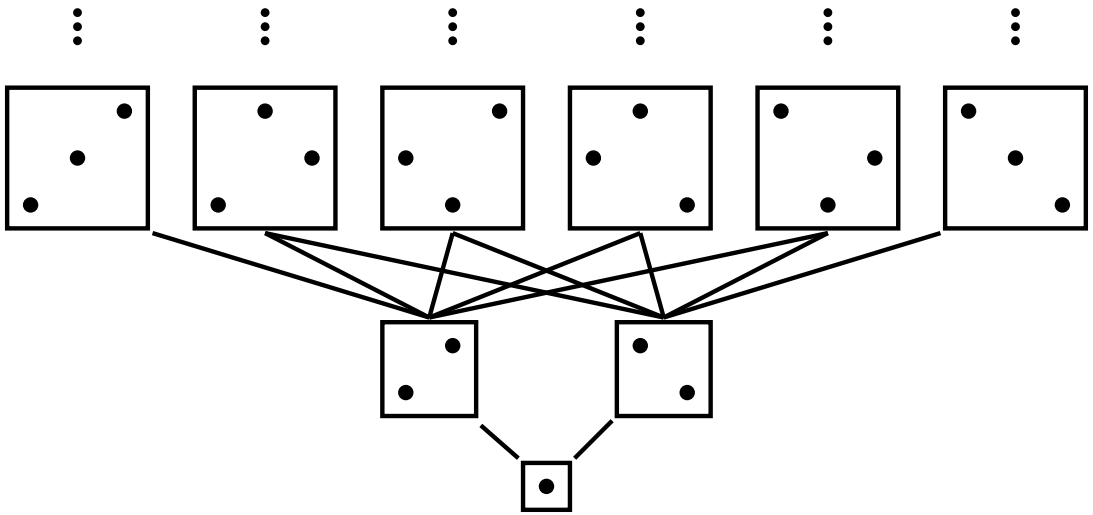
Permutation Poset



Permutation Poset



Permutation Poset



Permutation Poset

A *permutation class* is a downset in the permutation poset. In other words, if π is in the class $\mathcal{C}$ and $\sigma \leqslant \pi$, then we must have $\sigma \in \mathcal{C}$.

Permutation Poset

A *permutation class* is a downset in the permutation poset. In other words, if π is in the class $\mathcal{C}$ and $\sigma \leqslant \pi$, then we must have $\sigma \in \mathcal{C}$.

A class can be specified by the set of minimal permutations not in the class, called its *basis*.

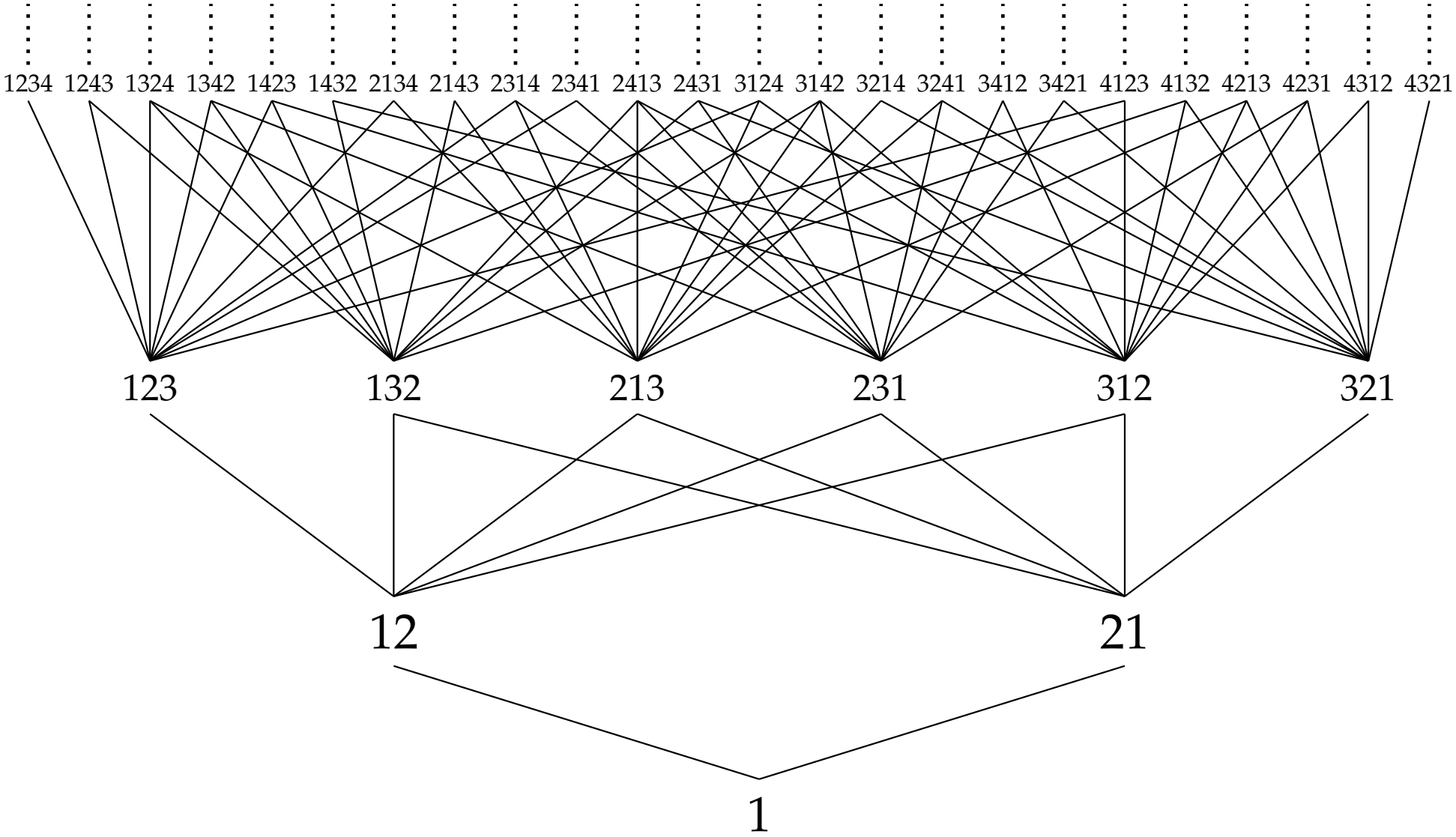
Permutation Poset

A *permutation class* is a downset in the permutation poset. In other words, if π is in the class $\mathcal{C}$ and $\sigma \leqslant \pi$, then we must have $\sigma \in \mathcal{C}$.

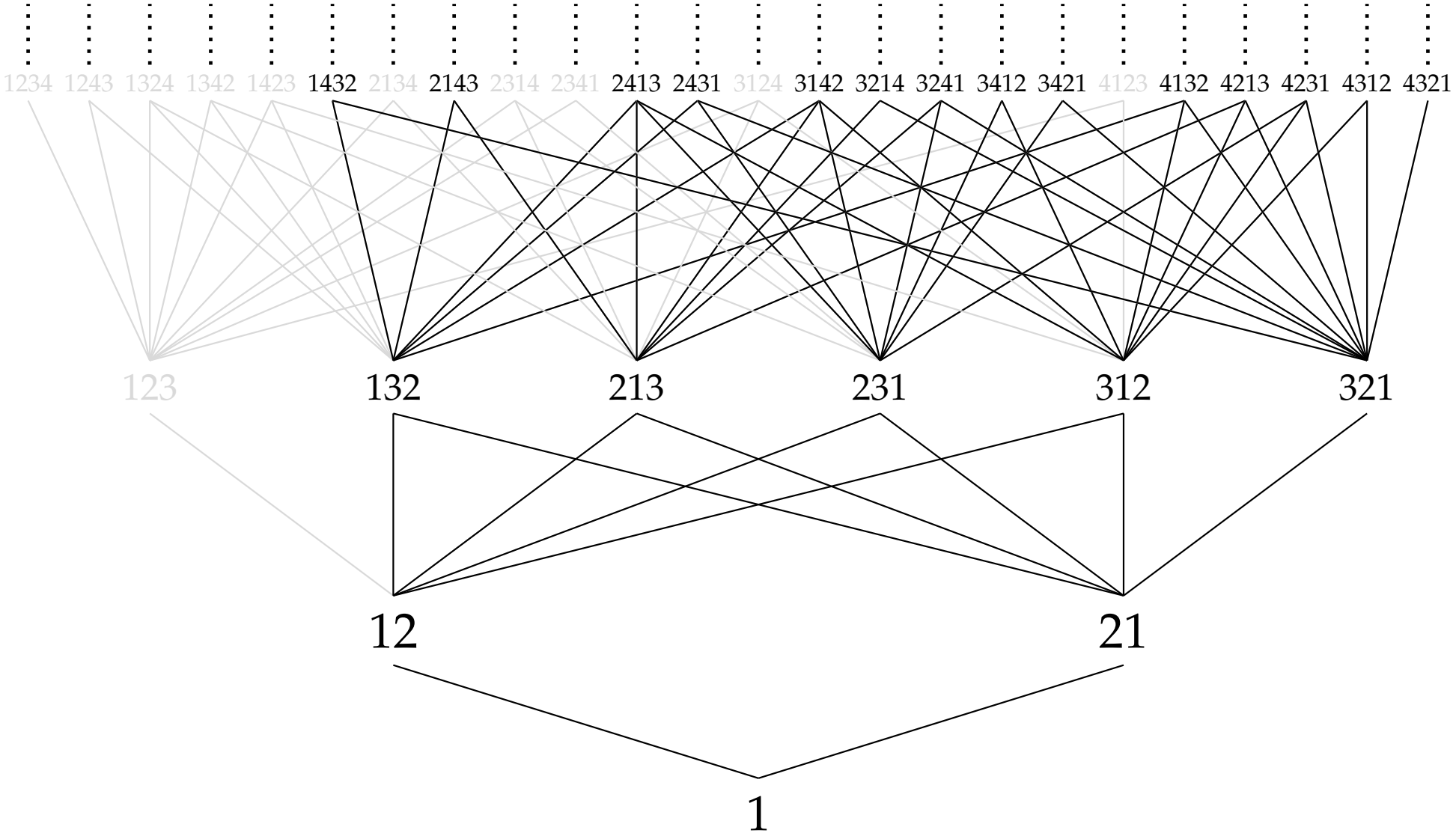
A class can be specified by the set of minimal permutations not in the class, called its *basis*.

The class with basis B is denoted $\mathrm{Av}(B)$.

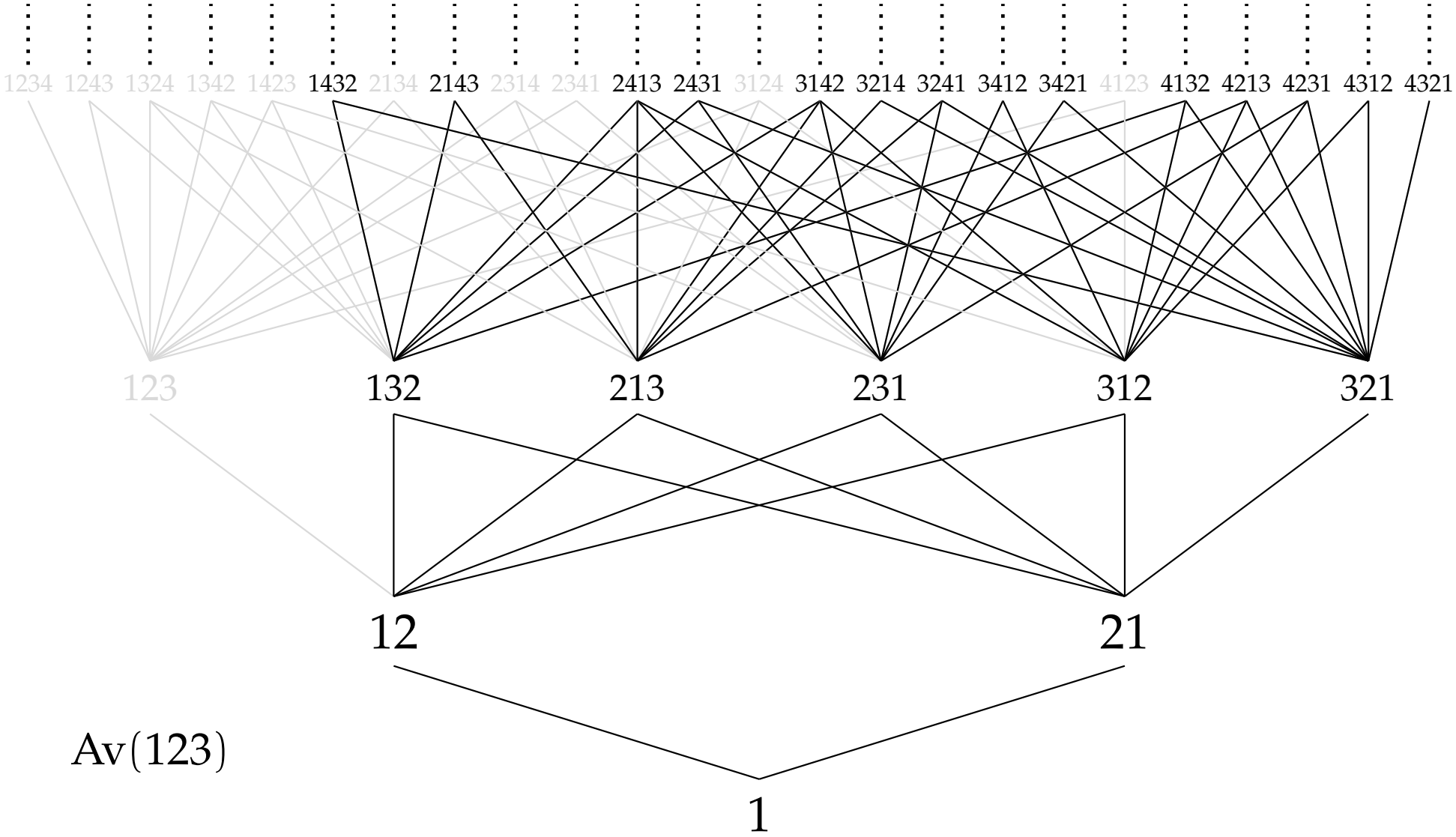
Permutation Poset



Permutation Poset



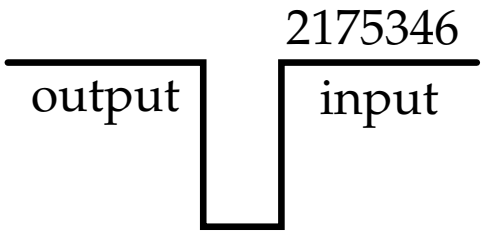
Permutation Poset



Av(123)

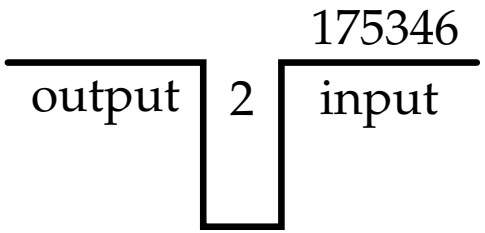
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



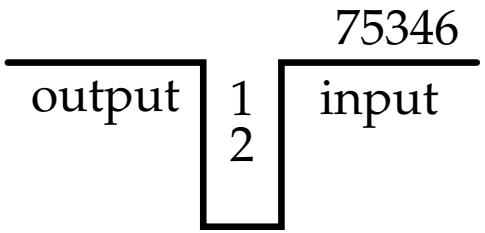
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



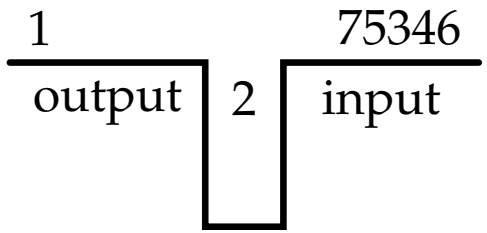
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



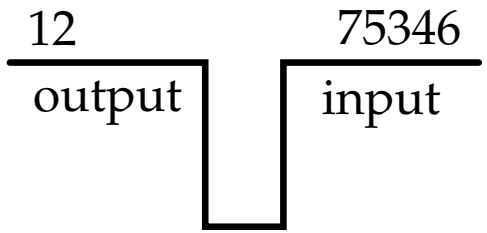
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



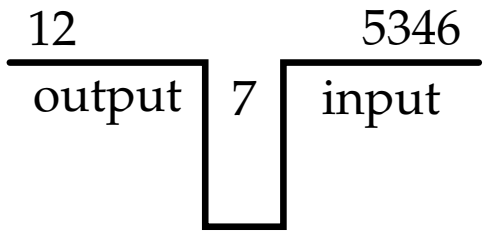
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



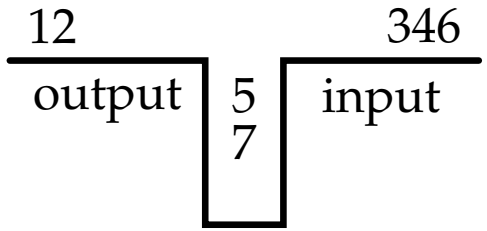
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



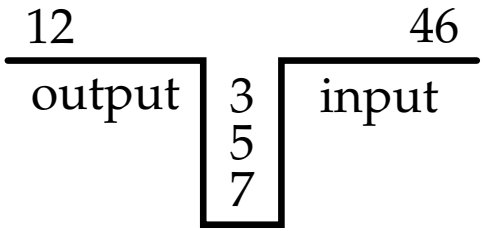
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



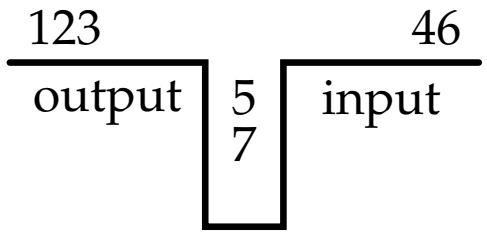
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



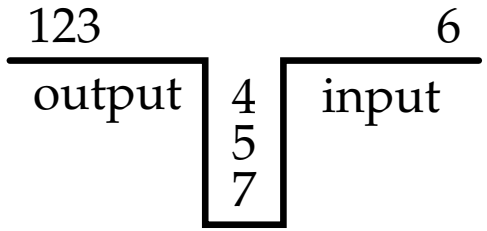
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



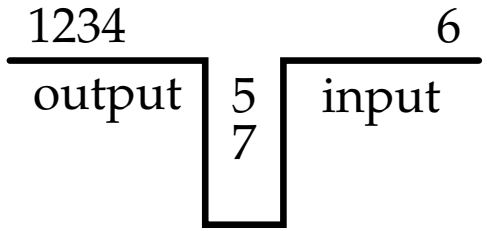
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



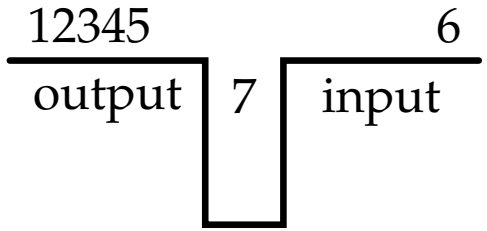
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



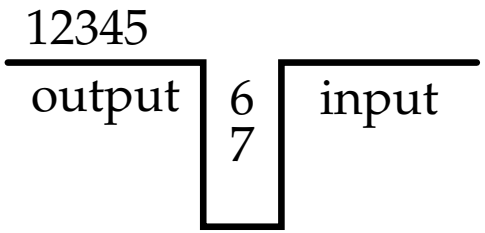
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



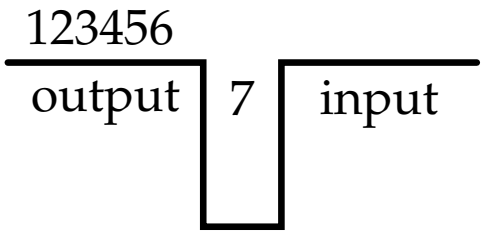
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



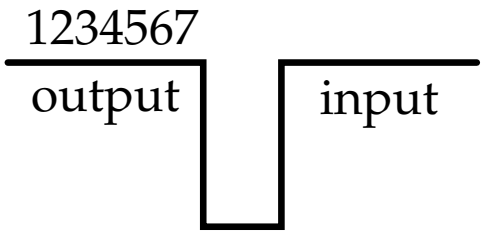
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



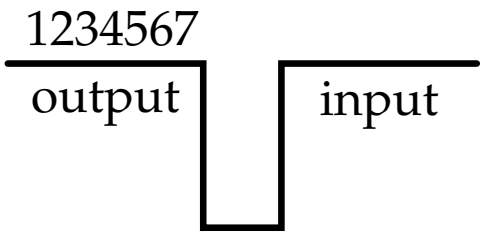
Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



Sorting with a Stack

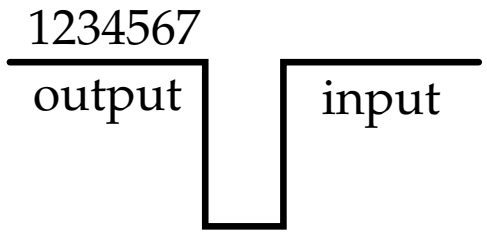
The permutation 2175346 can be sorted by a stack.



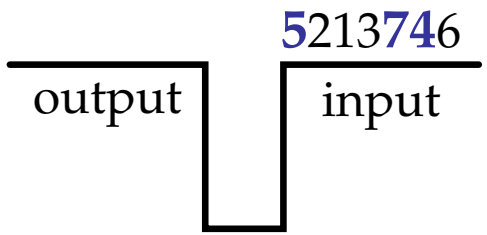
Not all permutations can be sorted by a stack.

Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

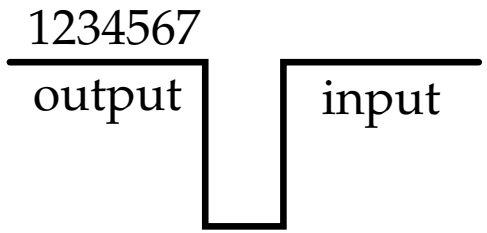


Not all permutations can be sorted by a stack.

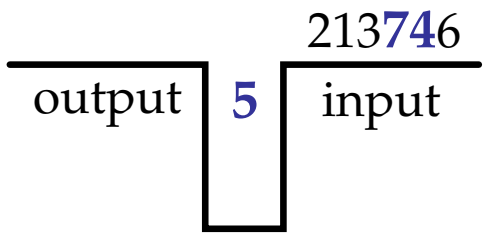


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

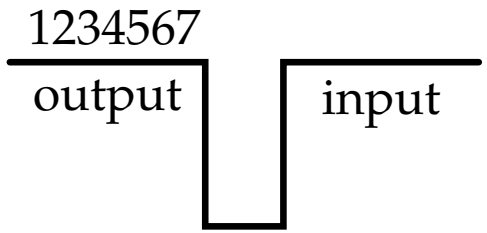


Not all permutations can be sorted by a stack.

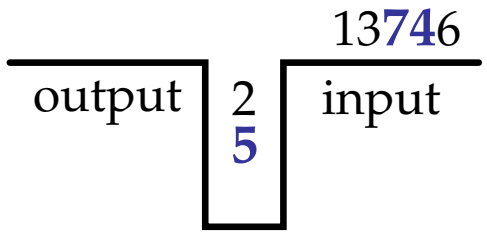


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

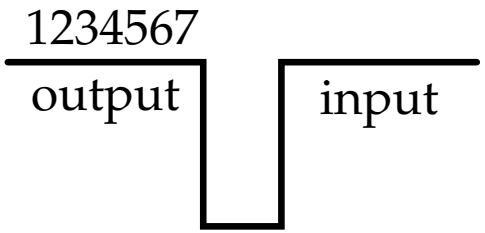


Not all permutations can be sorted by a stack.

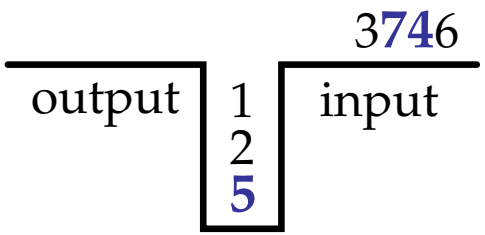


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

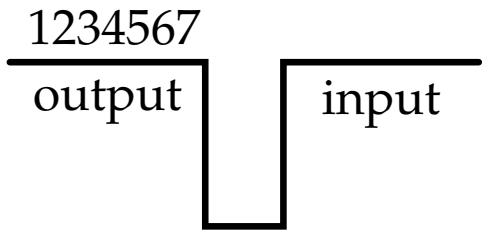


Not all permutations can be sorted by a stack.

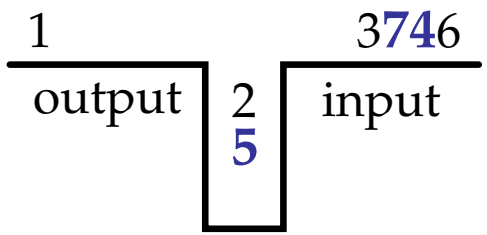


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

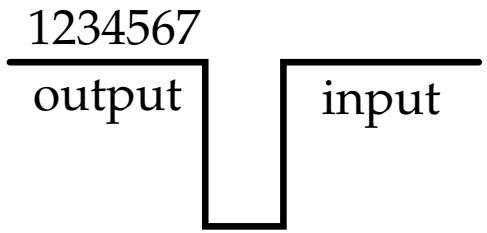


Not all permutations can be sorted by a stack.

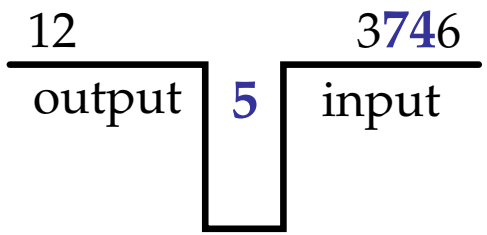


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

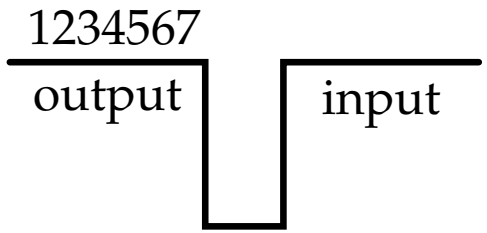


Not all permutations can be sorted by a stack.

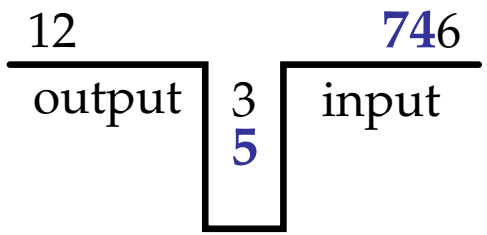


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

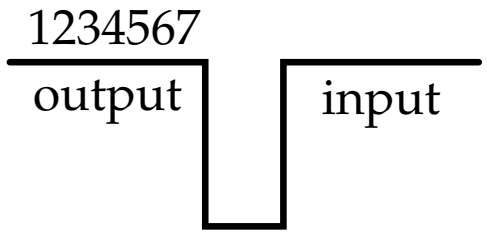


Not all permutations can be sorted by a stack.

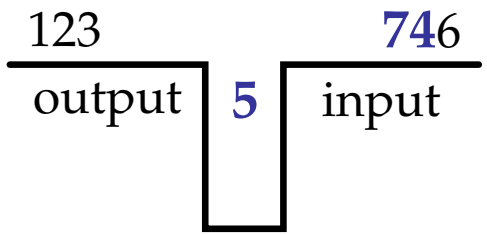


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

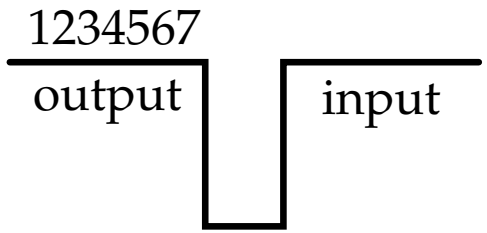


Not all permutations can be sorted by a stack.

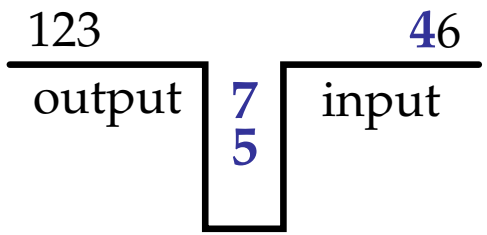


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

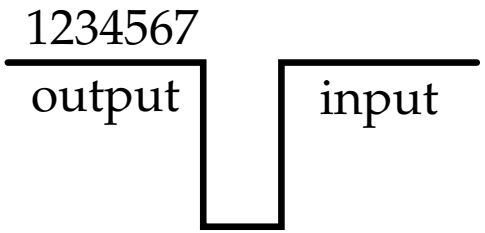


Not all permutations can be sorted by a stack.

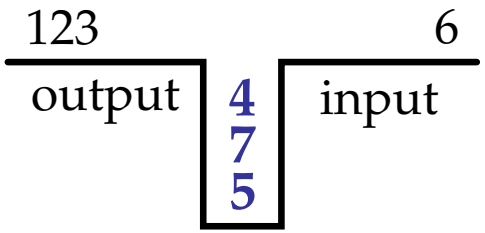


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

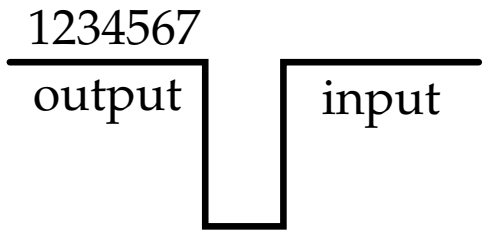


Not all permutations can be sorted by a stack.

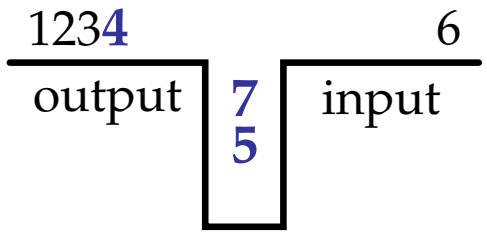


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

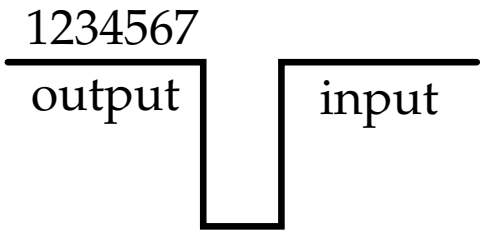


Not all permutations can be sorted by a stack.

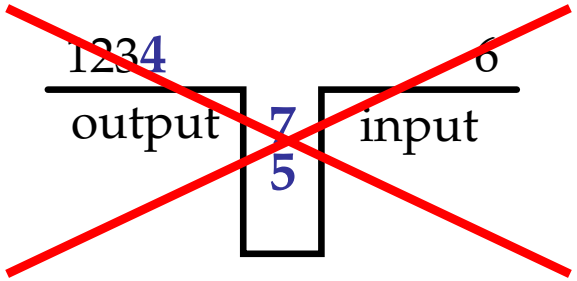


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.

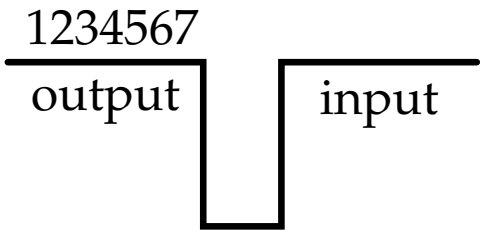


Not all permutations can be sorted by a stack.

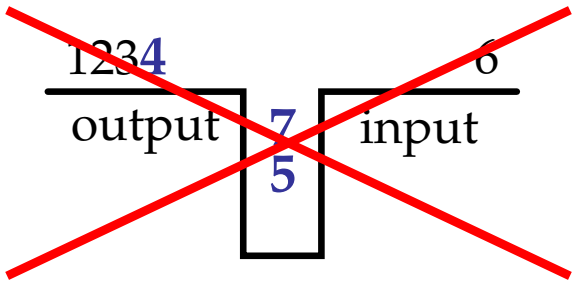


Sorting with a Stack

The permutation 2175346 can be sorted by a stack.



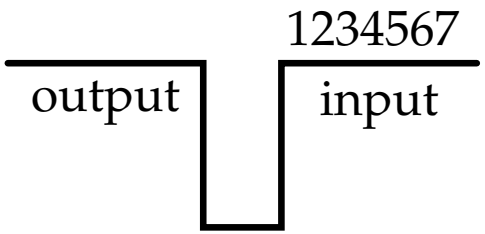
Not all permutations can be sorted by a stack.



Permutations containing 231 cannot be sorted by a stack. In fact, the permutations that are stack-sortable are exactly those in $\mathrm{Av}(231)$.

Generating with a Stack

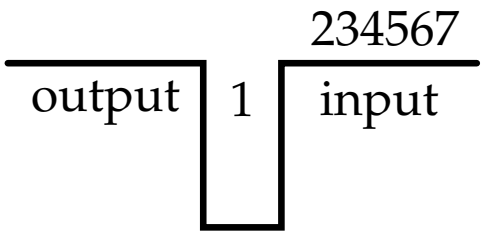
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

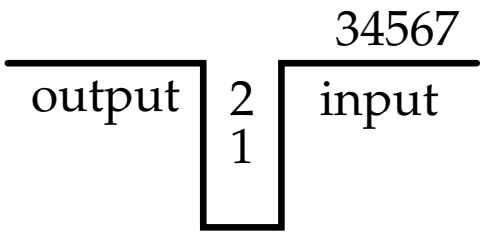
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

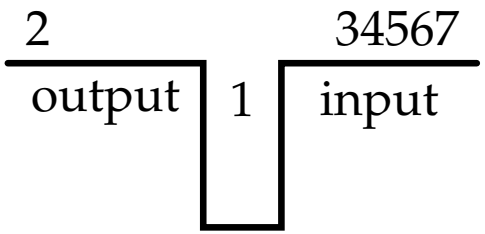
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

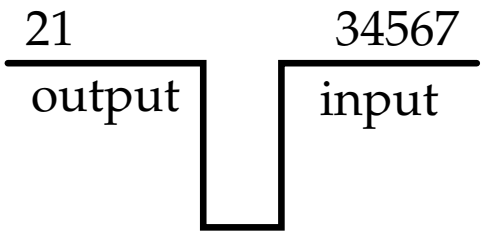
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

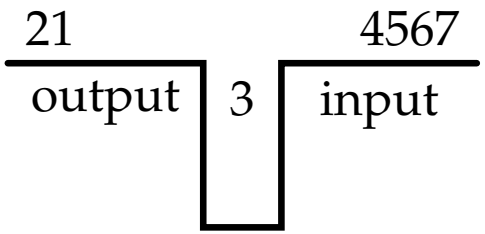
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

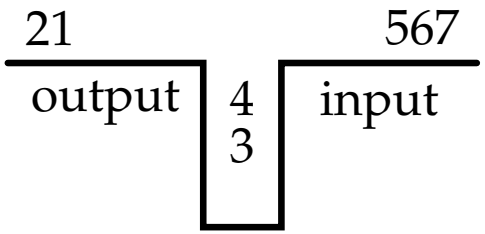
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

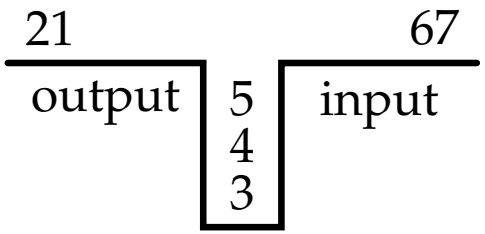
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

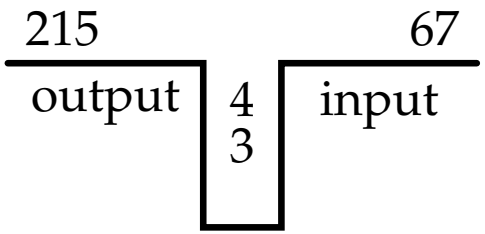
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

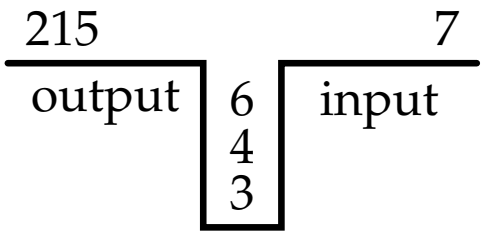
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

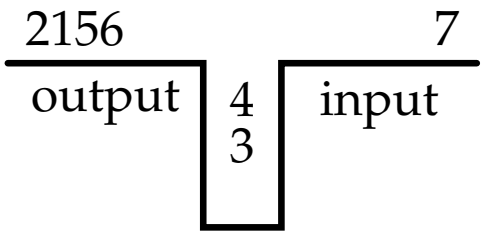
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

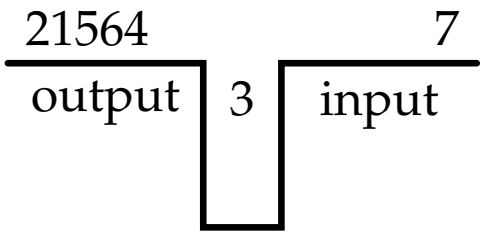
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

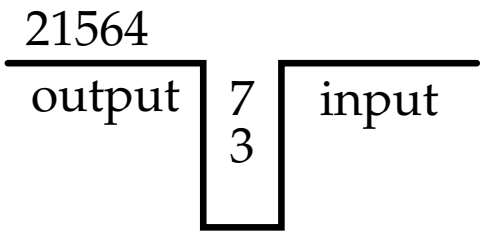
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

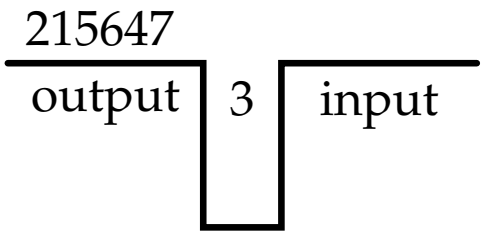
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

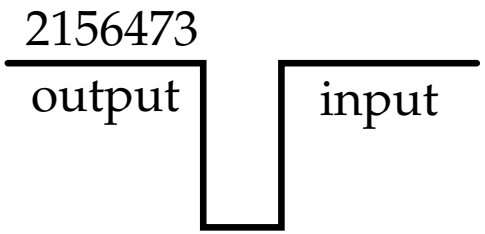
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

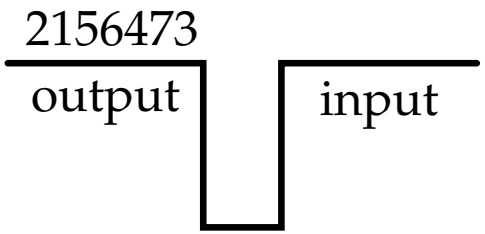
The permutation 2156473 can be generated by a stack.



(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

Generating with a Stack

The permutation 2156473 can be generated by a stack.

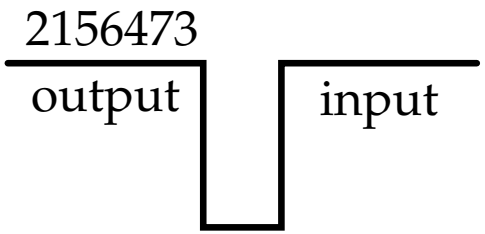


(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

The permutations that can be *generated* by a stack are exactly the inverses of those that can be *sorted* by a stack.

Generating with a Stack

The permutation 2156473 can be generated by a stack.



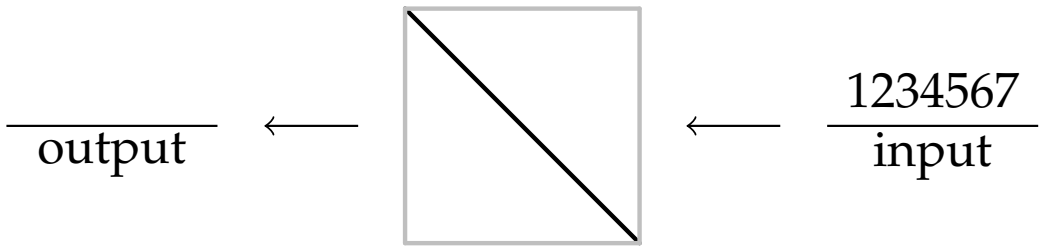
(Notice the stack always holds a decreasing subpermutation when read top to bottom.)

The permutations that can be *generated* by a stack are exactly the inverses of those that can be *sorted* by a stack.

A stack can generate the permutations in $\mathrm{Av}(231^{-1}) = \mathrm{Av}(312)$.

Generalizing Stacks

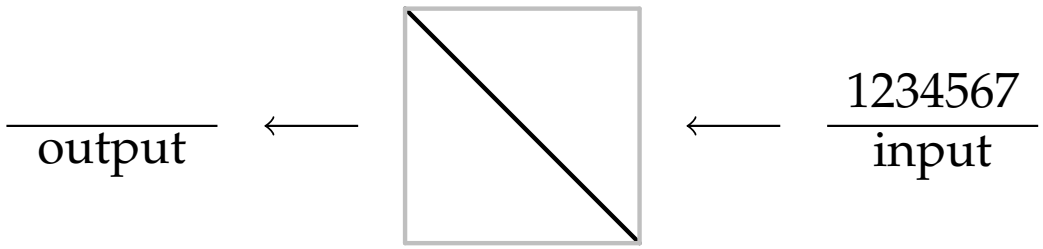
The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

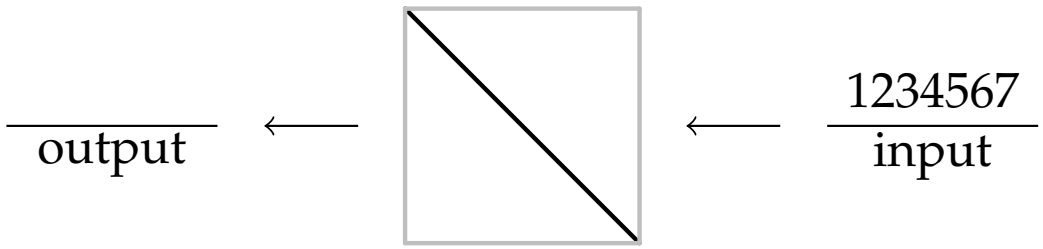
- push a new maximum entry into the container such that the container holds a decreasing permutation



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

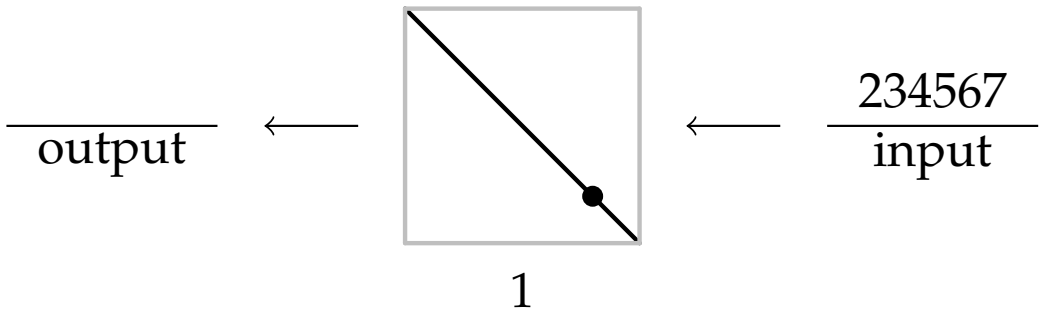
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

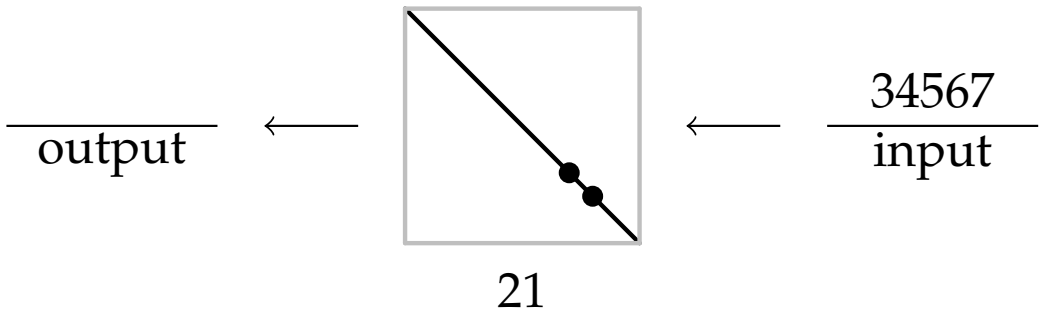
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

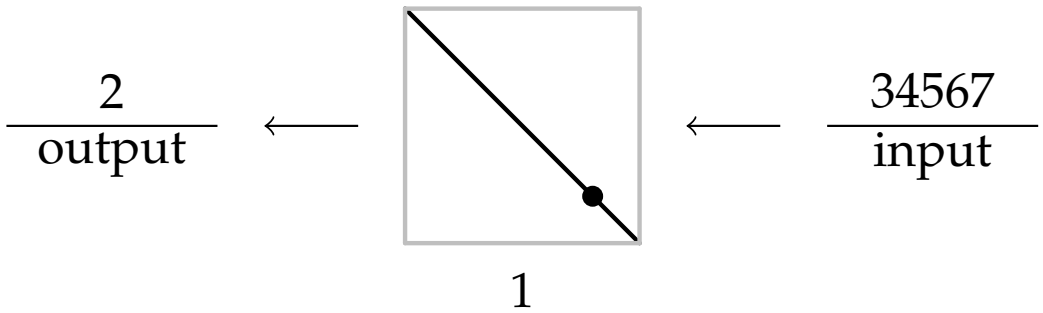
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

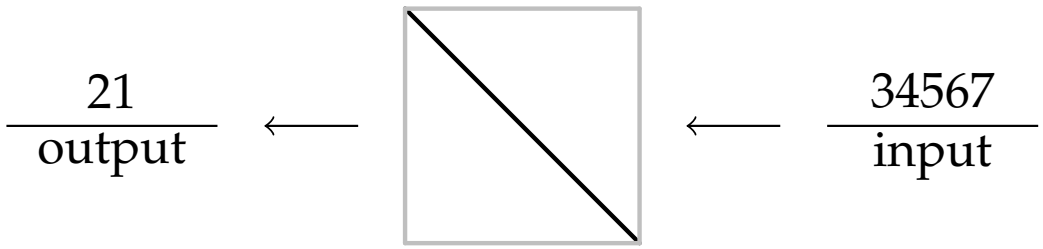
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

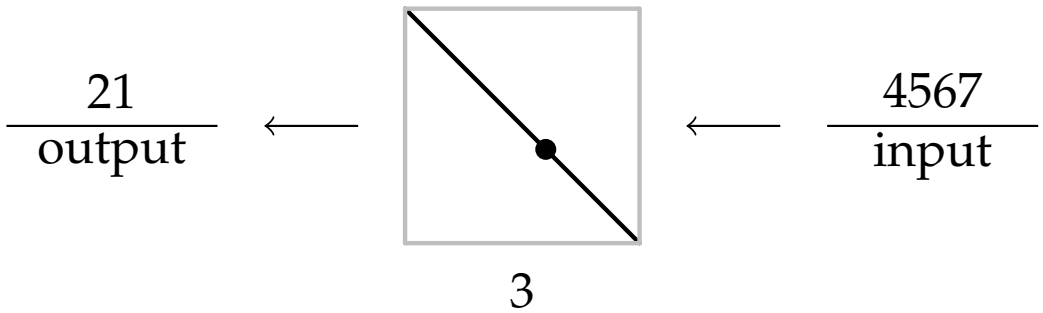
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

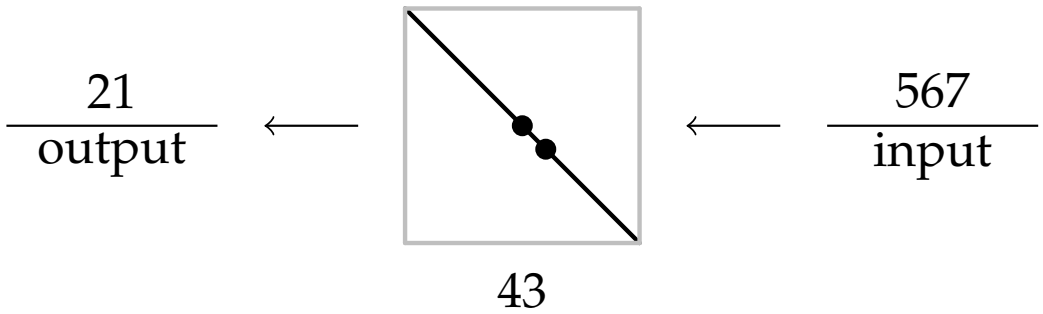
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

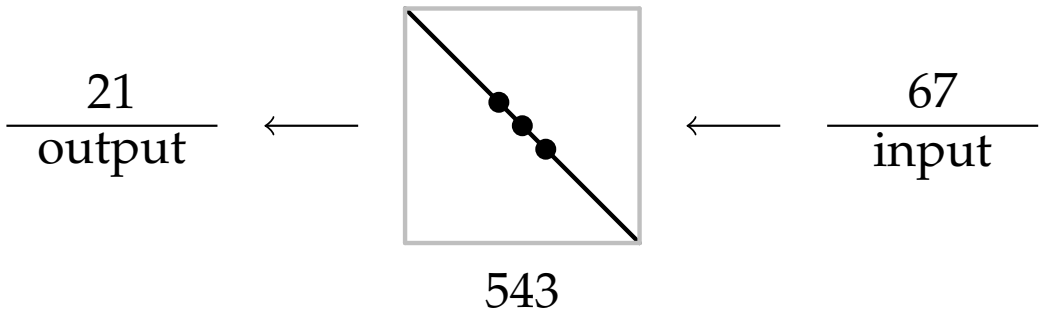
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

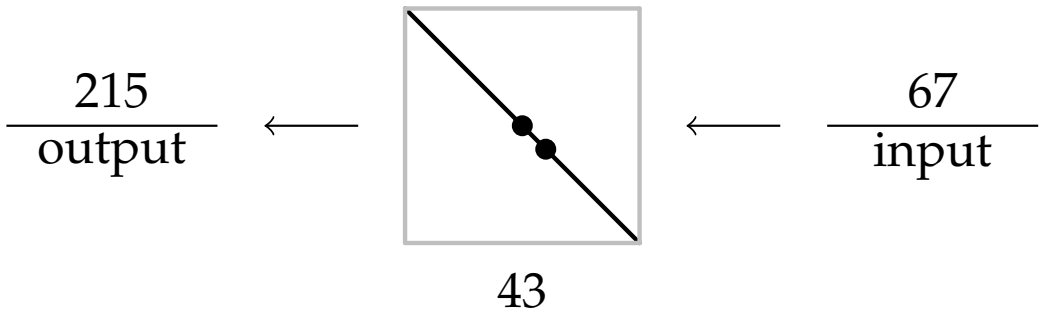
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

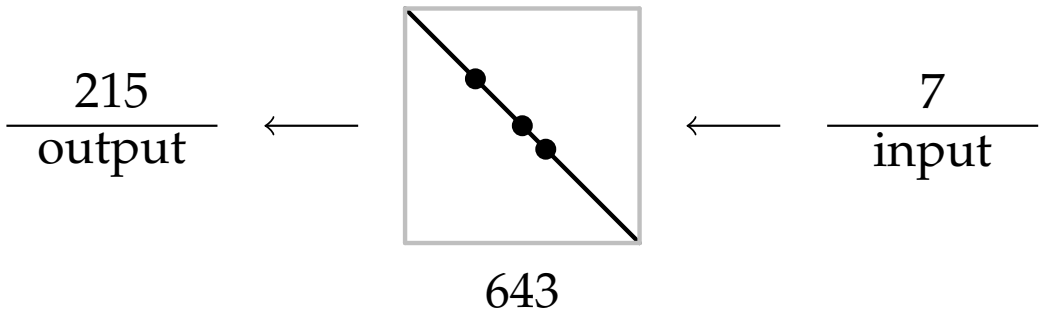
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

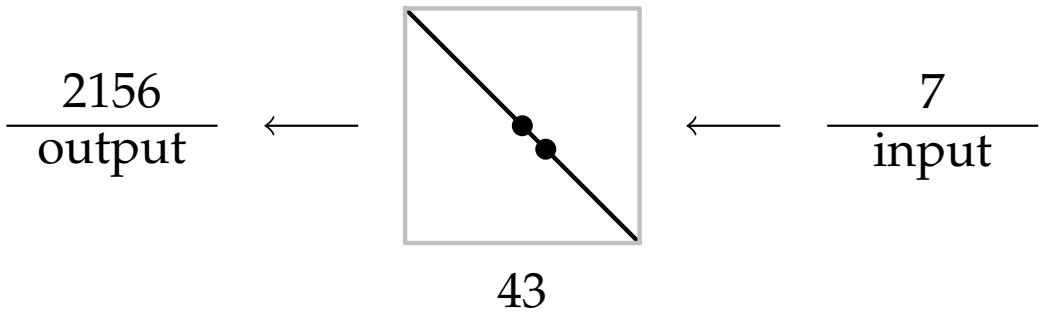
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

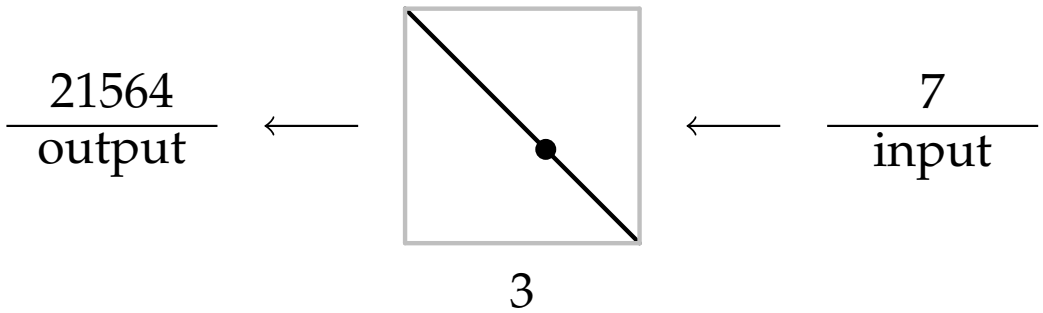
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

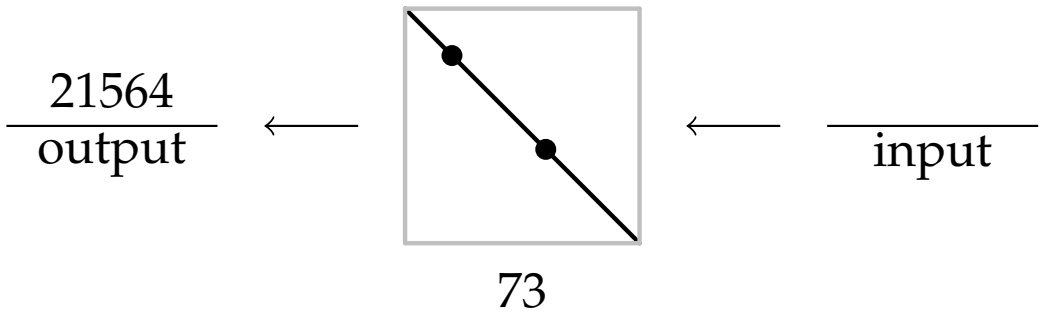
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

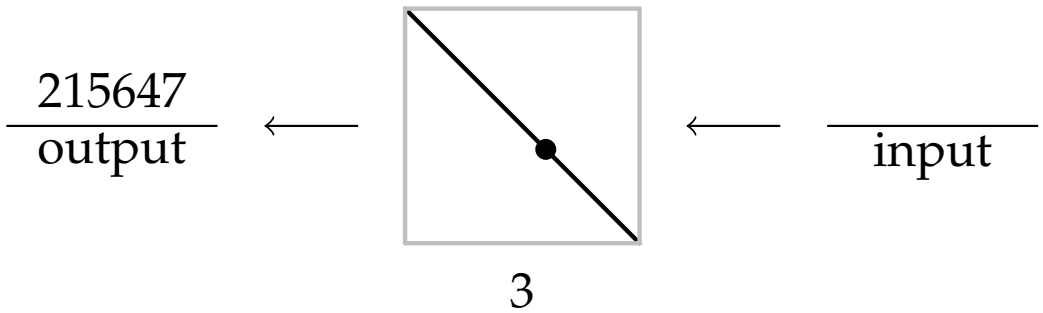
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

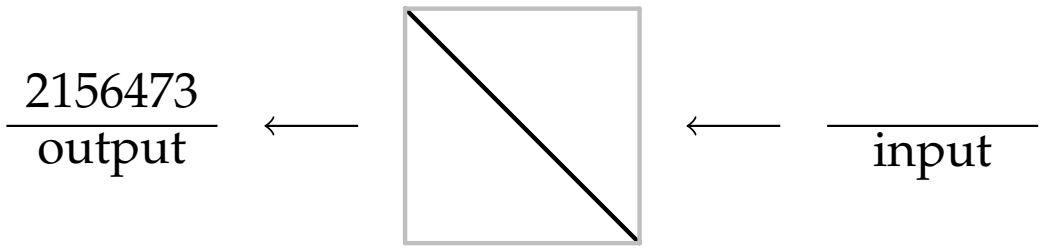
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

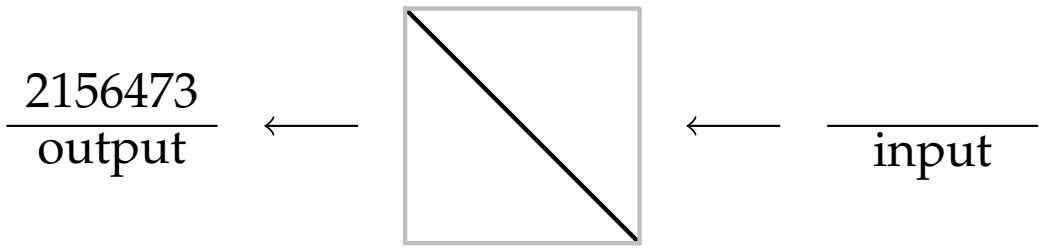
- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Generalizing Stacks

The stack can be thought of as a container that always holds a decreasing permutation, where at any time we can:

- push a new maximum entry into the container such that the container holds a decreasing permutation
- pop the leftmost entry out of the container.



Since the container holds permutations that avoid the pattern 12, we call this the Av(12)-machine.

Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

Generalizing Stacks

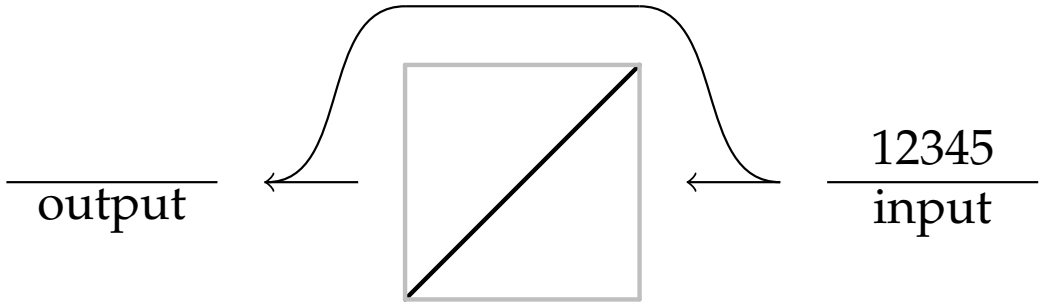
We allow a third operation: an entry can bypass the container and move straight from the input to the output.

In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.

Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

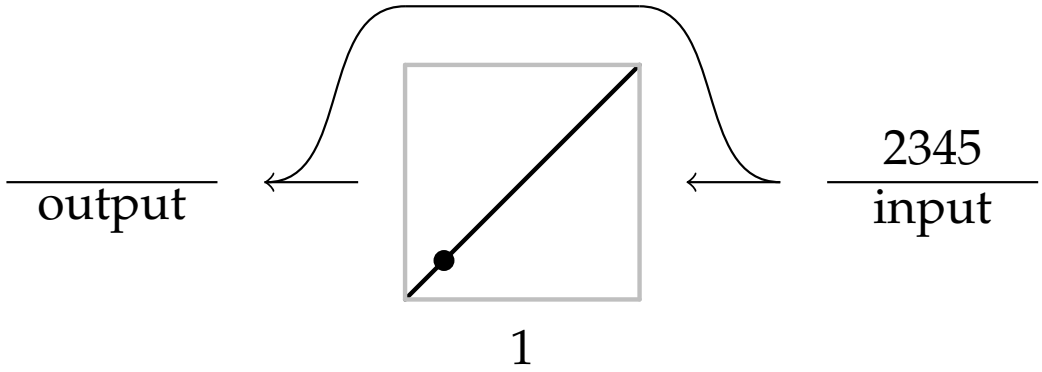
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

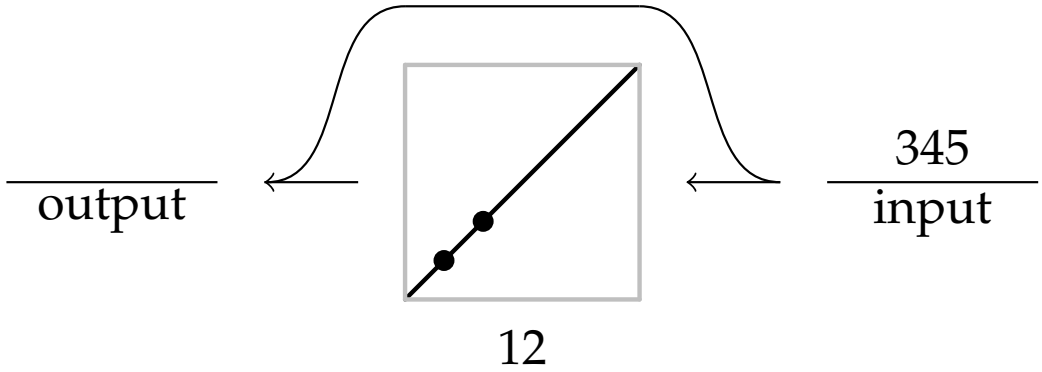
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

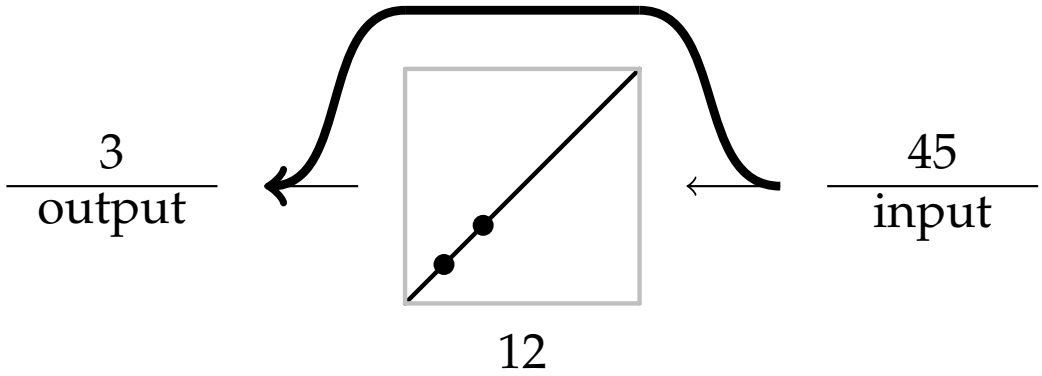
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

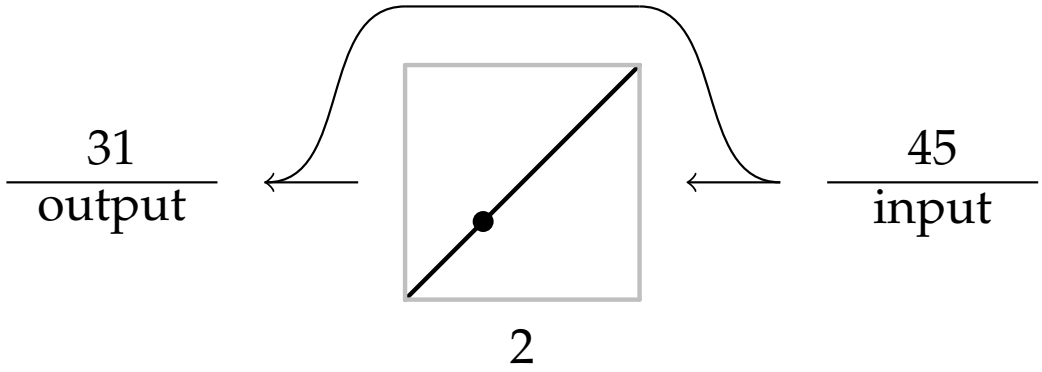
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

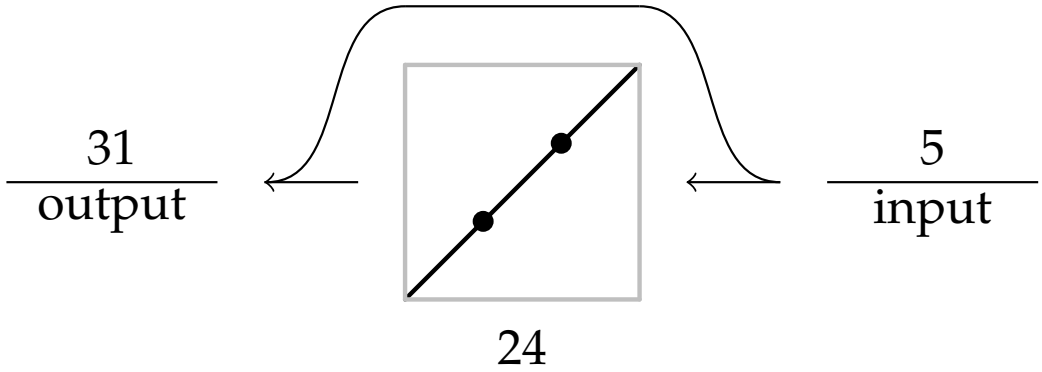
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

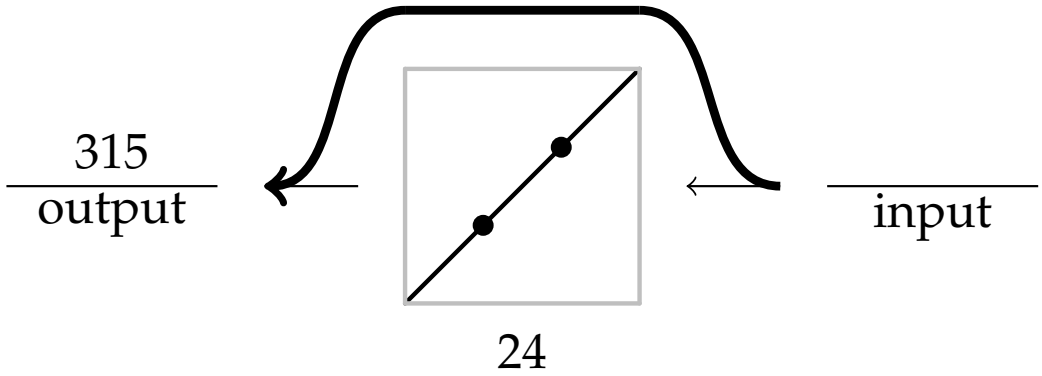
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

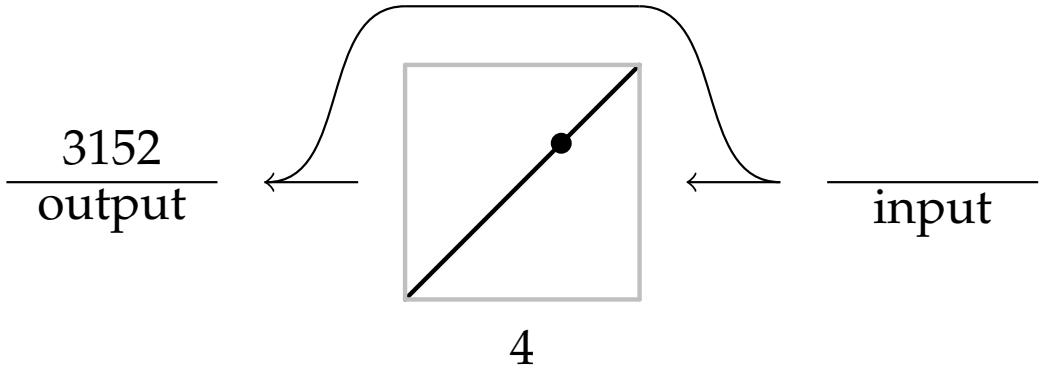
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

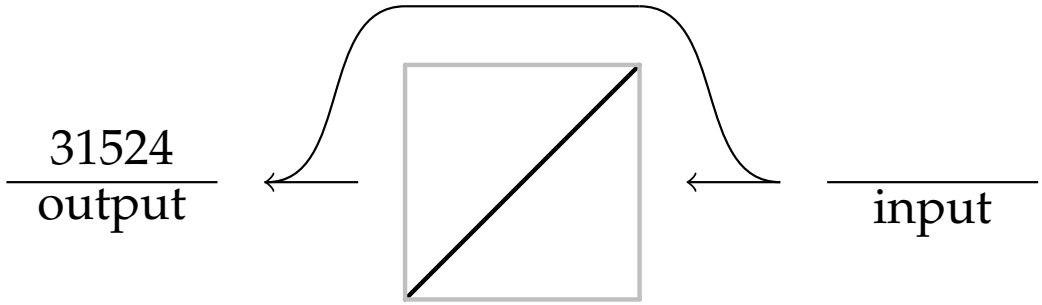
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

We allow a third operation: an entry can bypass the container and move straight from the input to the output.

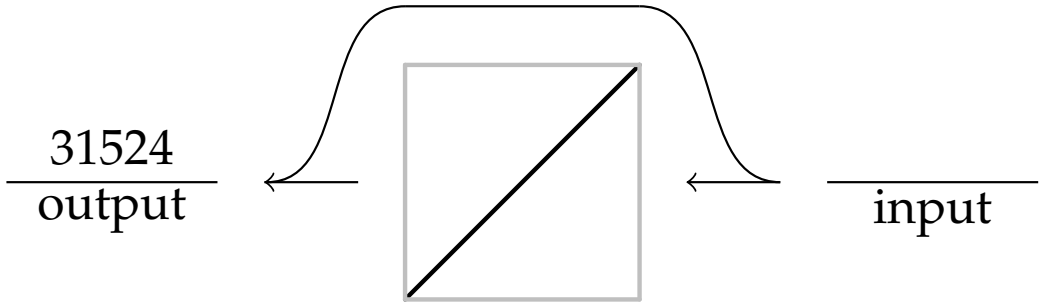
In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



Generalizing Stacks

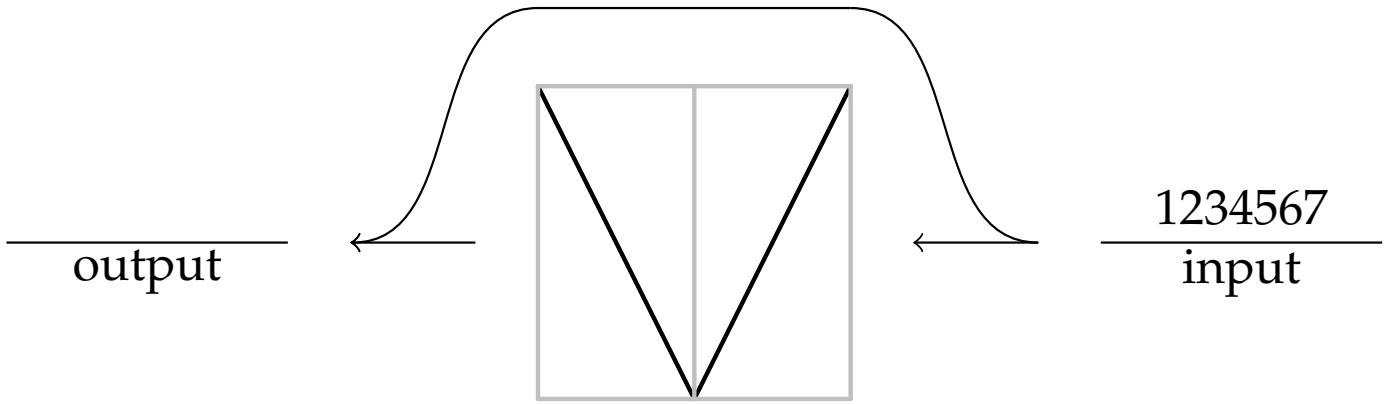
We allow a third operation: an entry can bypass the container and move straight from the input to the output.

In the Av(12)-machine, we didn't need the bypass because we could just push and then immediately pop.



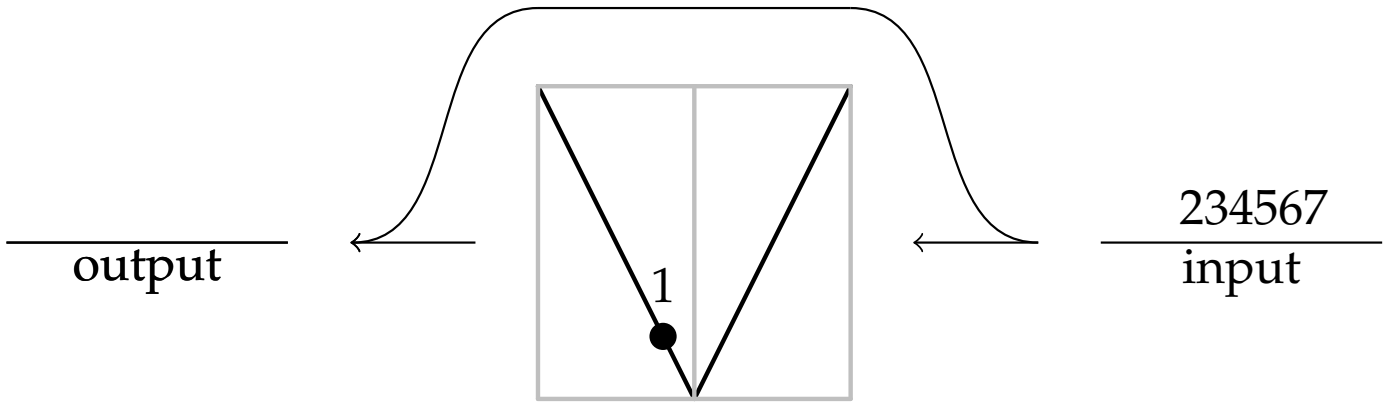
The Av(21)-machine generates the class Av(321).

Example: The Av(231, 132)-Machine



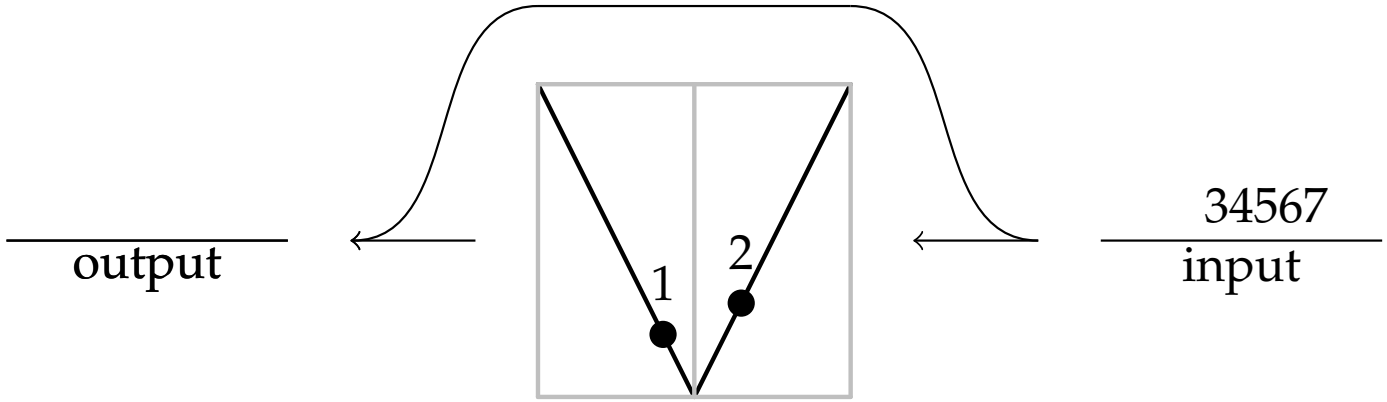
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



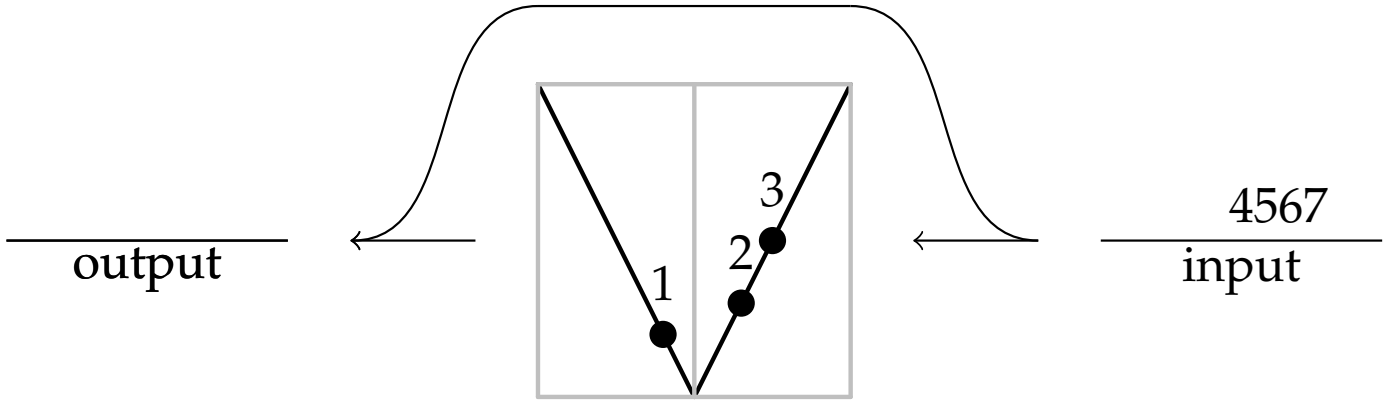
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



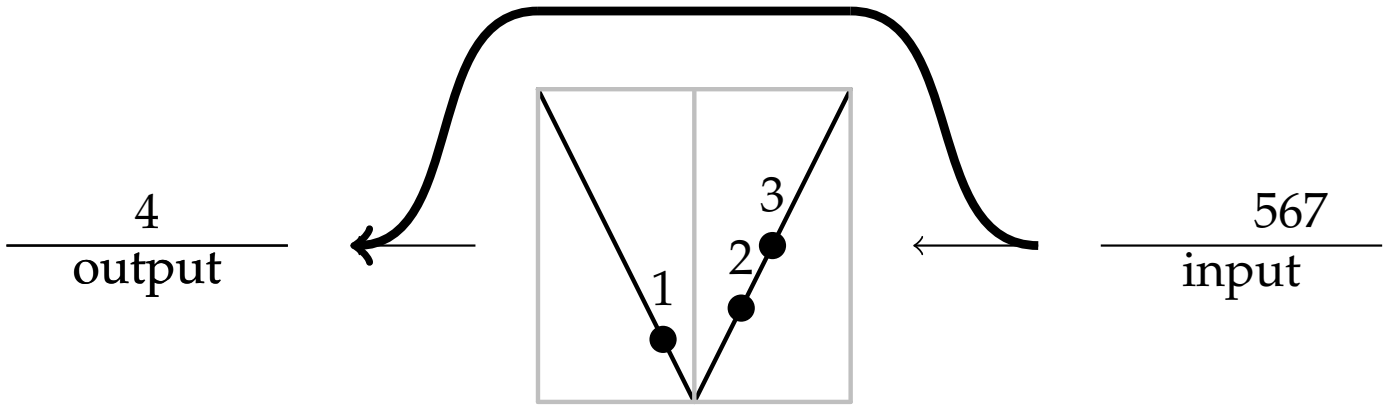
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



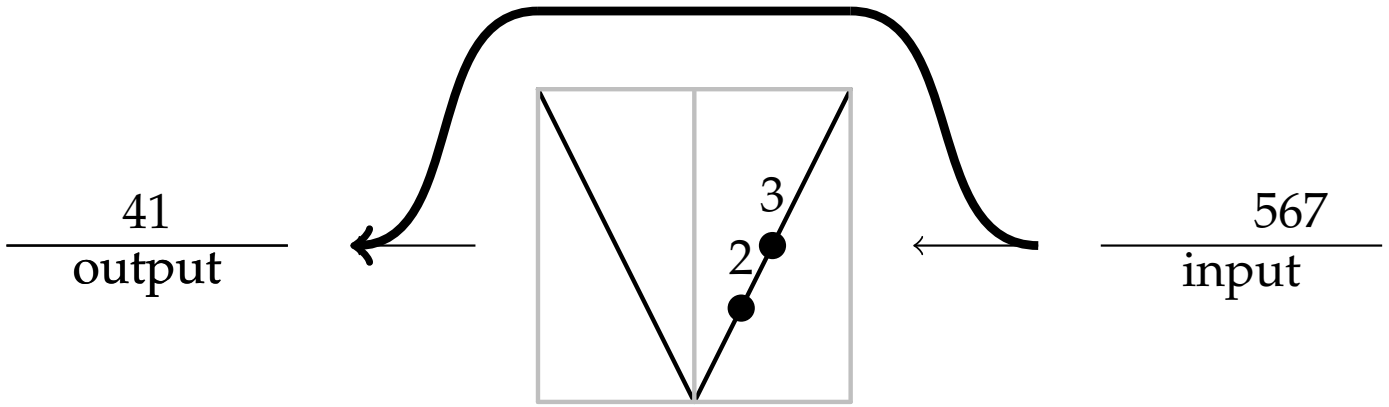
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



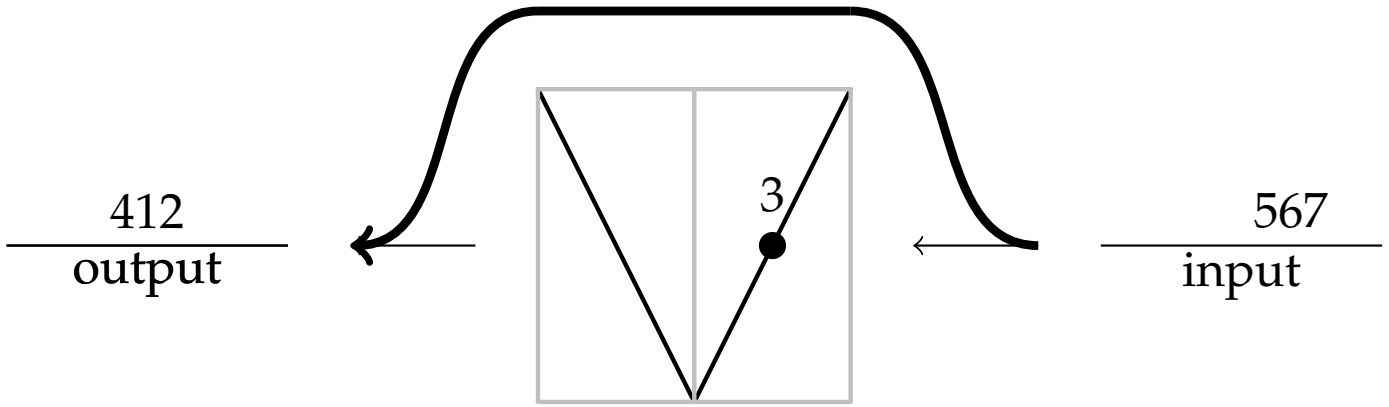
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



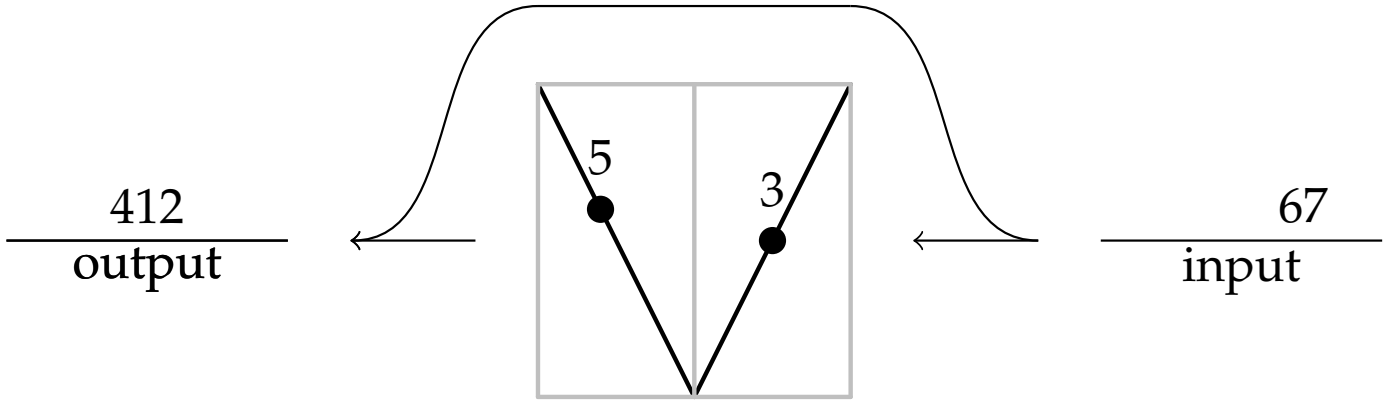
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



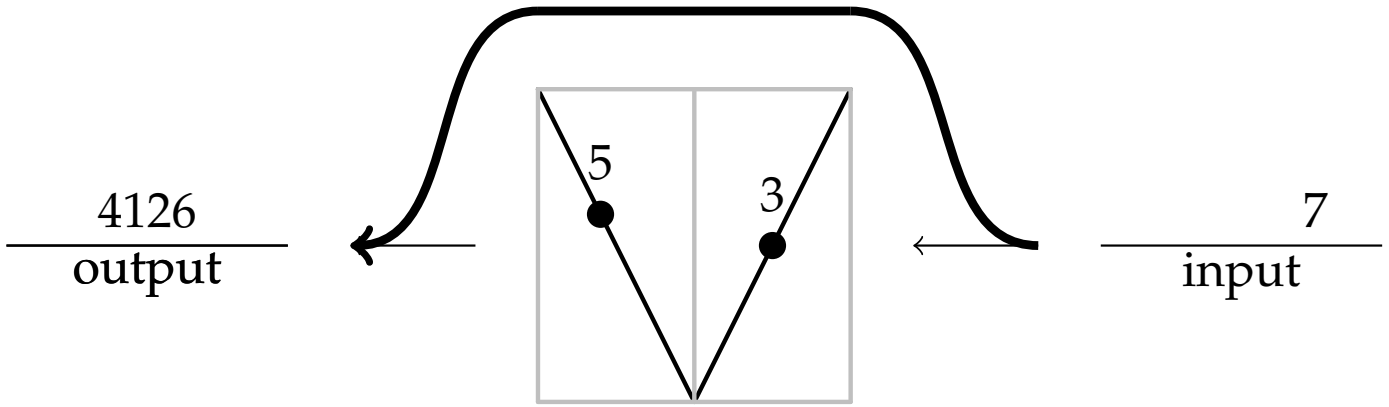
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



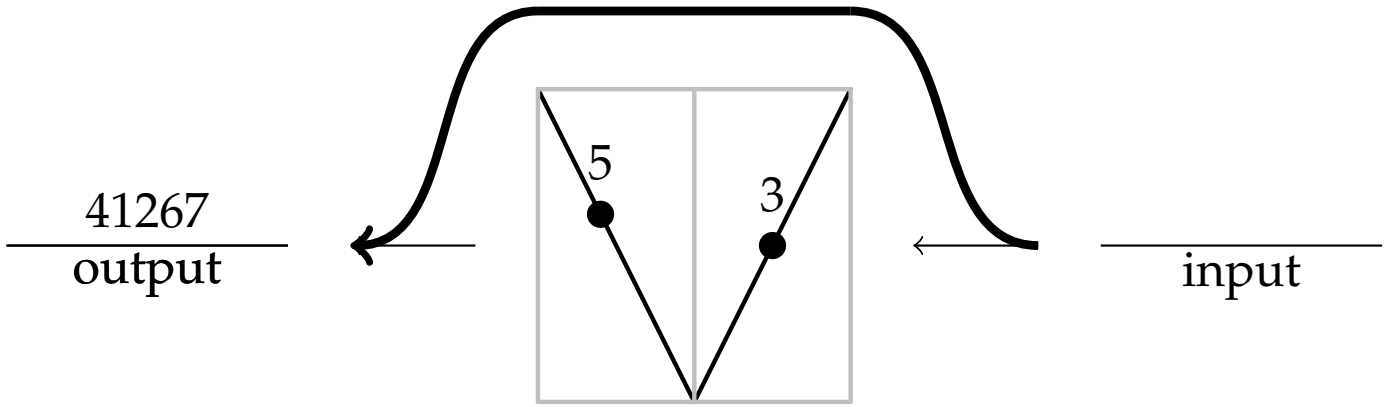
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



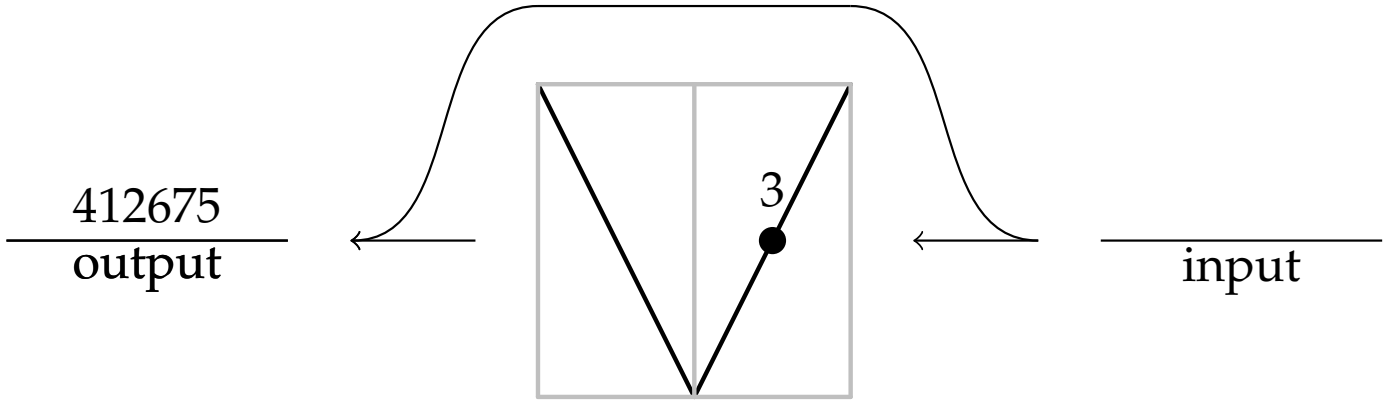
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



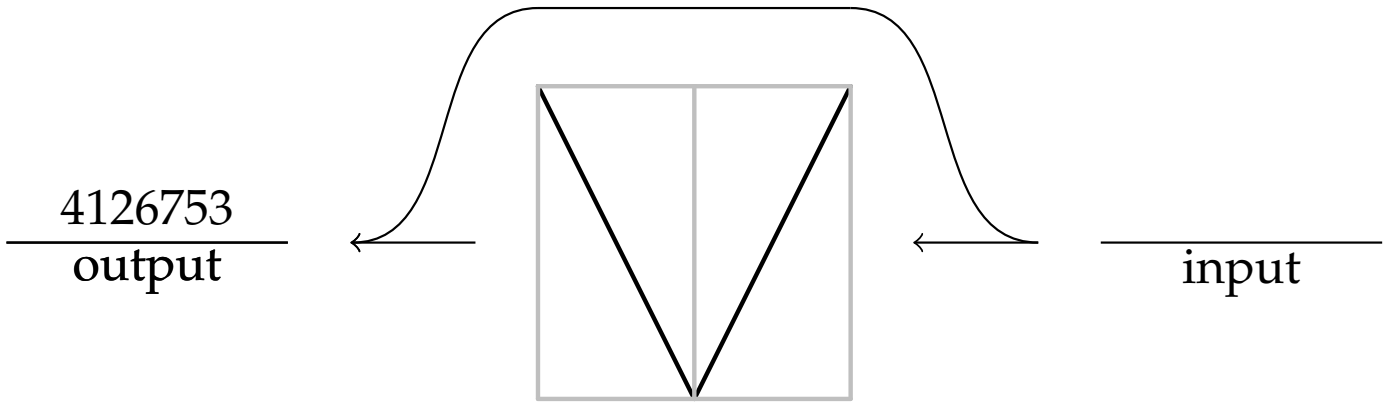
Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



Whenever the machine is nonempty, there are always two places to push a new entry.

Example: The Av(231, 132)-Machine



Whenever the machine is nonempty, there are always two places to push a new entry.

The Av(231, 132)-machine generates the class Av(4231, 4132).

The Basis Theorem

- The Av(12)-machine generates the class Av(312).

The Basis Theorem

- The Av(12)-machine generates the class Av(312).
- The Av(21)-machine generates the class Av(321).

The Basis Theorem

- The $\mathrm{Av}(12)$-machine generates the class $\mathrm{Av}(312)$.
- The $\mathrm{Av}(21)$-machine generates the class $\mathrm{Av}(321)$.
- The $\mathrm{Av}(231, 132)$-machine generates the class $\mathrm{Av}(4231, 4132)$.

The Basis Theorem

- The $\mathrm{Av}(12)$-machine generates the class $\mathrm{Av}(312)$.
- The $\mathrm{Av}(21)$-machine generates the class $\mathrm{Av}(321)$.
- The $\mathrm{Av}(231, 132)$-machine generates the class $\mathrm{Av}(4231, 4132)$.

Theorem. The $\mathrm{Av}(B)$-machine generates the class

$$\mathrm{Av}(\{{}^{+}\beta : \beta \in B\}),$$

where ${}^{+}\beta$ is formed by adding a new maximum in front of β.

The Basis Theorem

- The $\mathrm{Av}(12)$-machine generates the class $\mathrm{Av}(312)$.
- The $\mathrm{Av}(21)$-machine generates the class $\mathrm{Av}(321)$.
- The $\mathrm{Av}(231, 132)$-machine generates the class $\mathrm{Av}(4231, 4132)$.

Theorem. The $\mathrm{Av}(B)$-machine generates the class

$$\mathrm{Av}(\{{}^{+}\beta : \beta \in B\}),$$

where ${}^{+}\beta$ is formed by adding a new maximum in front of β.

Example: The $\mathrm{Av}(123, 4132)$-machine generates the class $\mathrm{Av}(4123, 54132)$.

What do you get?

- **Structural Descriptions:**
  Every permutation in $\mathcal{C}$ can be formed by ...

What do you get?

- **Structural Descriptions:**
  Every permutation in $\mathcal{C}$ can be formed by ...
- **More efficient counting algorithms**

What do you get?

- **Structural Descriptions:**
  Every permutation in $\mathcal{C}$ can be formed by ...
- **More efficient counting algorithms**
- **Random sampling**

What do you get?

- **Structural Descriptions:**
  Every permutation in $\mathcal{C}$ can be formed by ...
- **More efficient counting algorithms**
- **Random sampling**
- **Bijections**

Enumerating Av(312) via the Av(12)-Machine

Av(312): $\approx 4^n$

Av(12): $\approx n$

Enumerating Av(312) via the Av(12)-Machine

$\mathrm{Av}(312)$: $\approx 4^n$

$\mathrm{Av}(12)$: $\approx n$

Instead of remembering all $\approx 4^n$ permutations of length n in $\mathrm{Av}(312)$ to build those of length $n+1$, we only need to remember the $n+1$ possible states of the machine:

[machine has no entries],
[machine has 1 entry],
...,
[machine has n entries]

Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

Enumerating Av(312) via the Av(12)-Machine

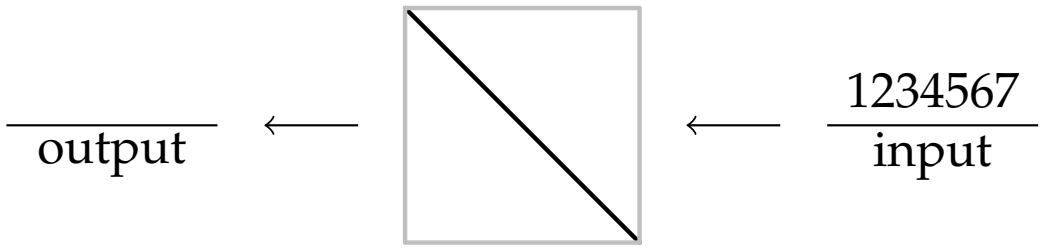
Even better, from the Av(12)-machine we can find the generating function for Av(312).

Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.

Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

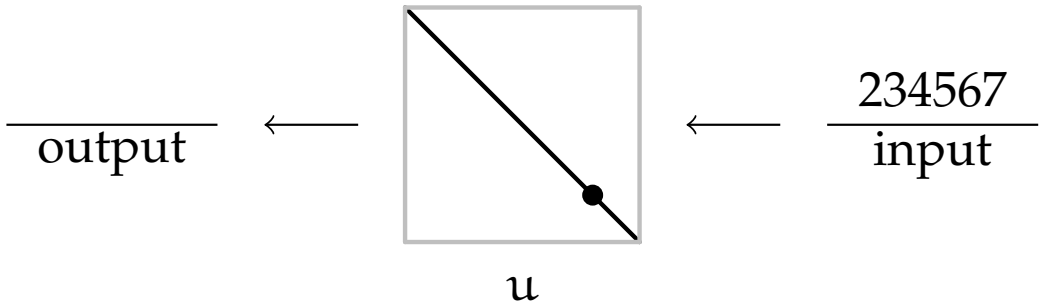
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

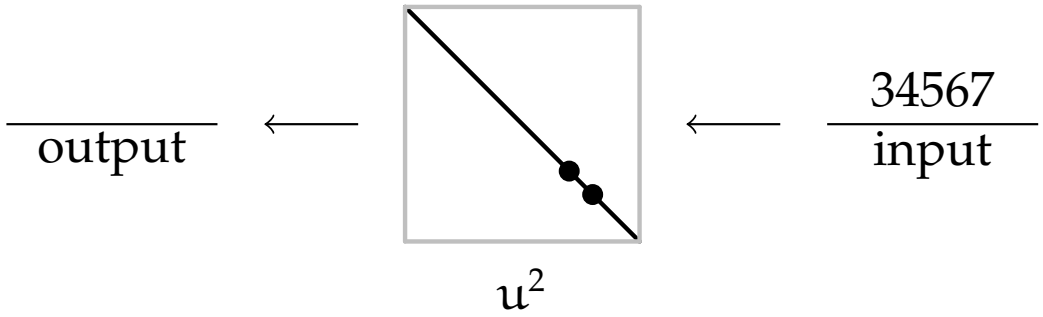
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

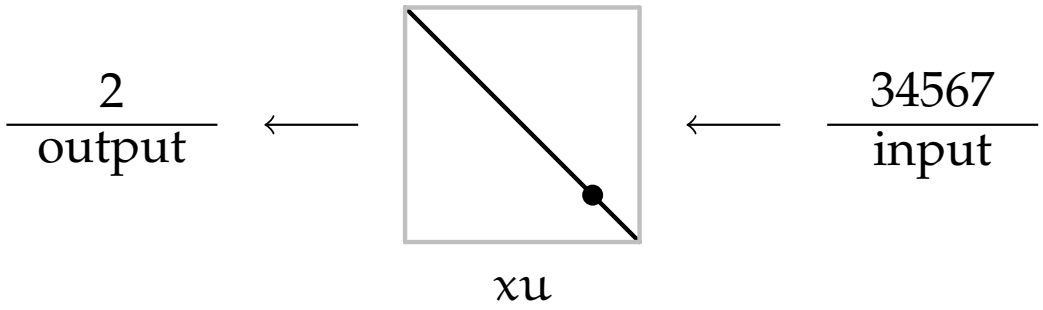
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

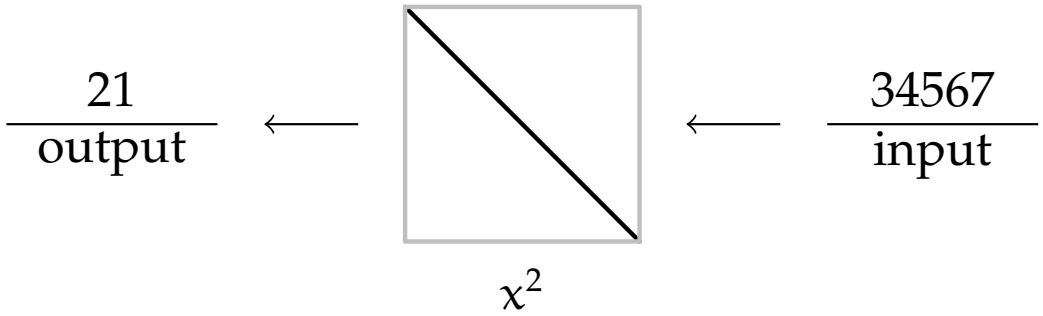
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

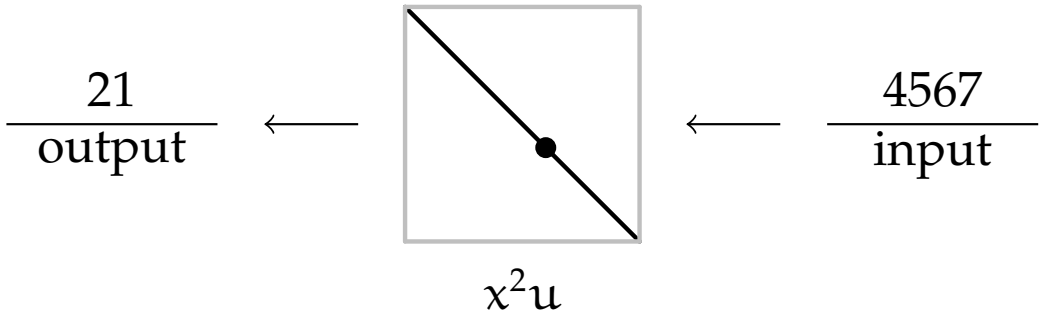
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

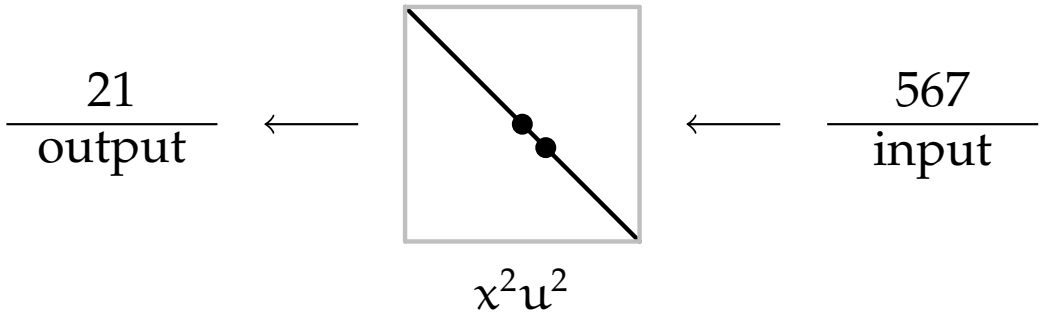
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

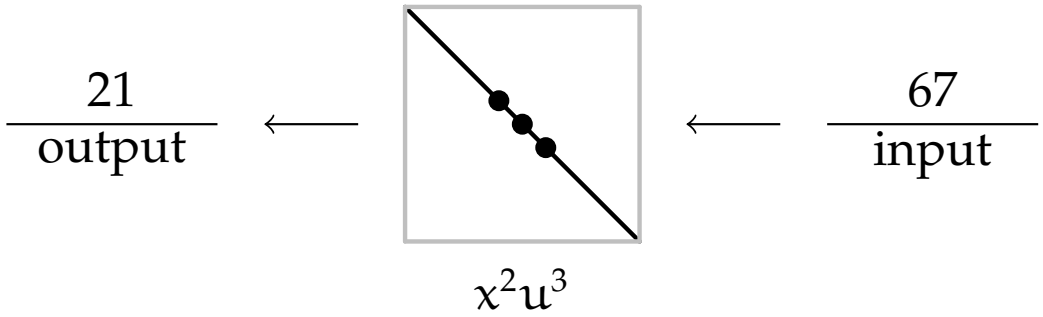
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

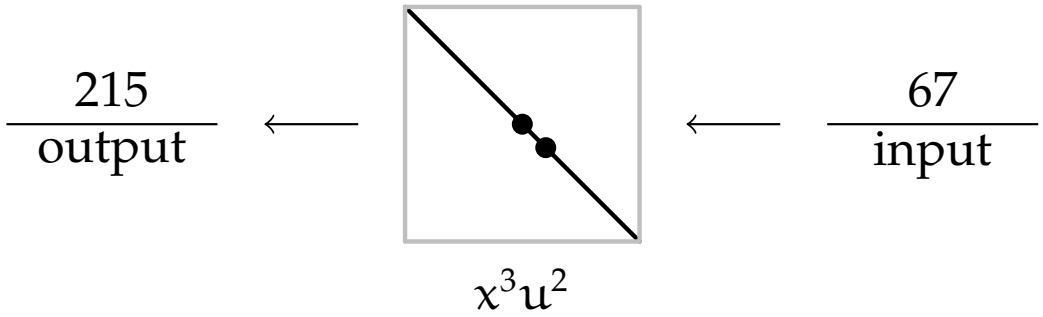
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

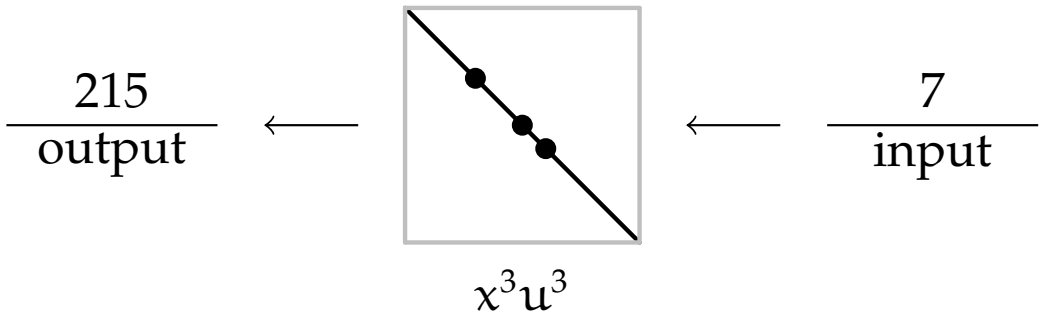
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

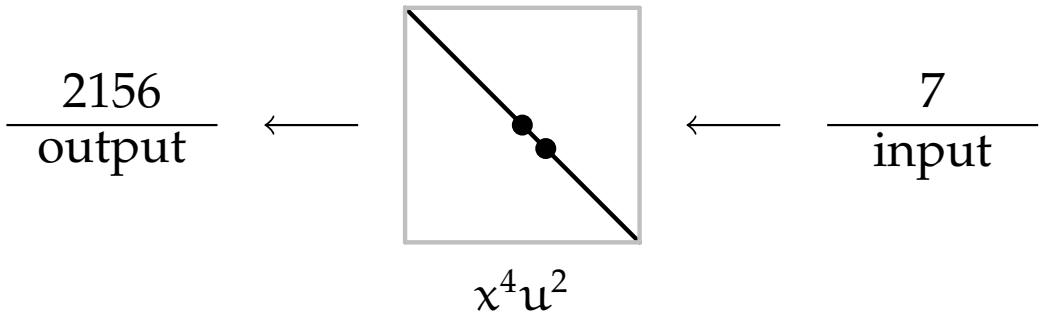
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

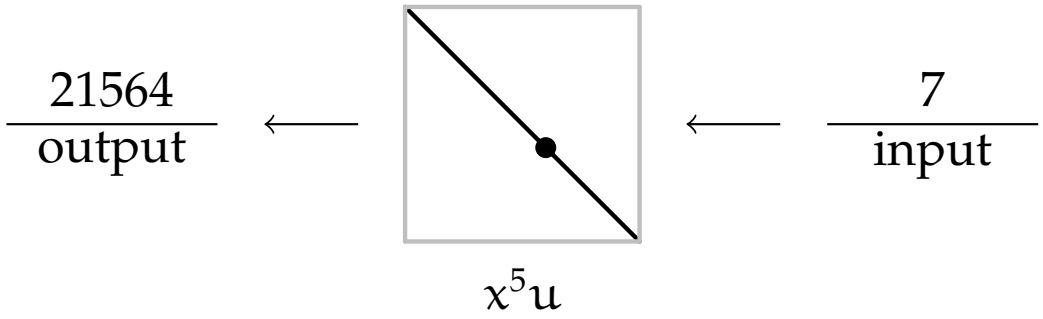
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

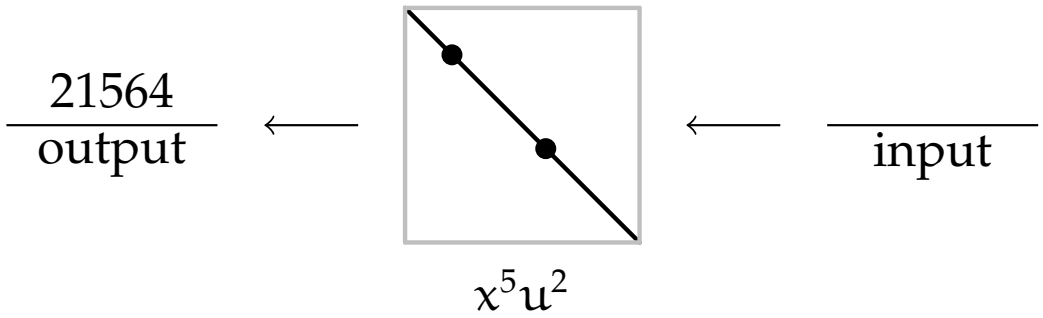
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

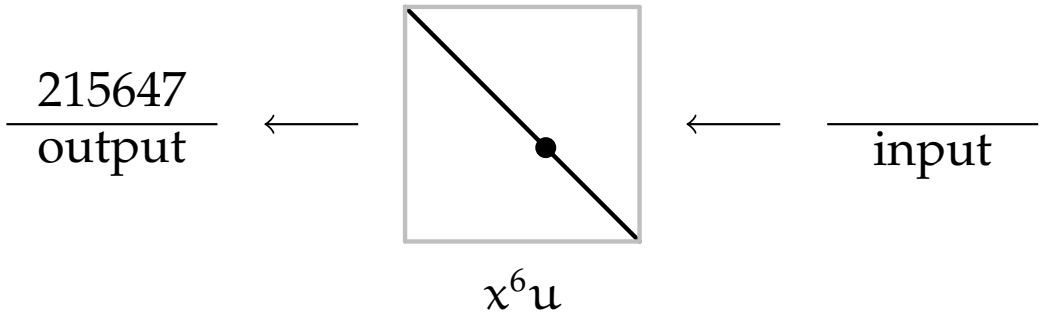
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

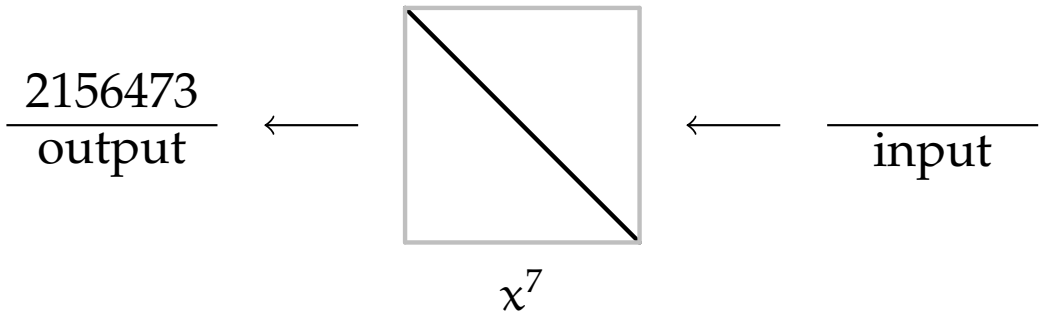
Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.

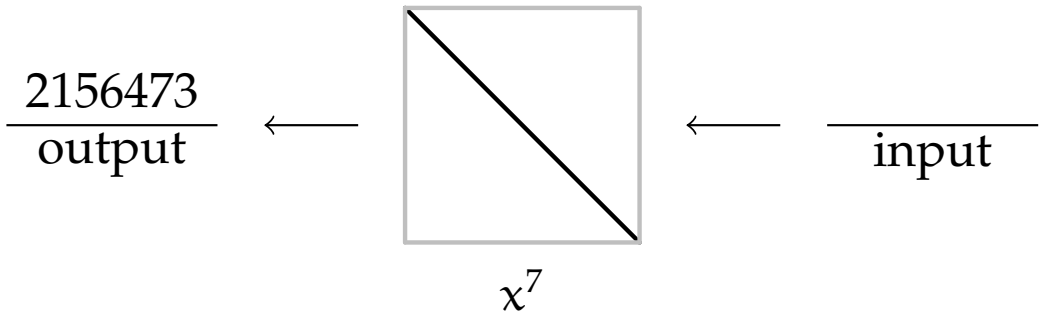


States with no entries in the machine are those with no u term.

Enumerating Av(312) via the Av(12)-Machine

Even better, from the Av(12)-machine we can find the generating function for Av(312).

Let $f(x, u)$ be the generating function for valid states of the Av(12)-machine where u tracks the number of entries in the machine and x tracks the number that have been output so far.



States with no entries in the machine are those with no u term.

Hence the generating function for these states is $f(x, 0)$.

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state
- from pushing an entry into a previous state

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state
- from pushing an entry into a previous state
- from popping an entry from a *nonempty* state

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state
- from pushing an entry into a previous state
- from popping an entry from a *nonempty* state

This translates to a functional equation:

$$f(x,u) = 1 + uf(x,u) + \frac{x}{u}\left(f(x,u) - f(x,0)\right)$$

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state
- from pushing an entry into a previous state
- from popping an entry from a *nonempty* state

This translates to a functional equation:

$$\mathbf{f(x,u)} = 1 + uf(x,u) + \frac{x}{u}\left(f(x,u) - f(x,0)\right)$$

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- **as the start state**
- from pushing an entry into a previous state
- from popping an entry from a *nonempty* state

This translates to a functional equation:

$$f(x,u) = \mathbf{1} + uf(x,u) + \frac{x}{u}\left(f(x,u) - f(x,0)\right)$$

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state
- **from pushing an entry into a previous state**
- from popping an entry from a *nonempty* state

This translates to a functional equation:

$$f(x,u) = 1 + \mathbf{uf(x,u)} + \frac{x}{u}\left(f(x,u) - f(x,0)\right)$$

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state
- from pushing an entry into a previous state
- **from popping an entry from a *nonempty* state**

This translates to a functional equation:

$$f(x,u) = 1 + uf(x,u) + \frac{x}{u}\left(f(x,u) - f(x,0)\right)$$

Enumerating Av(312) via the Av(12)-Machine

Every machine state arises either:

- as the start state
- from pushing an entry into a previous state
- from popping an entry from a *nonempty* state

This translates to a functional equation:

$$f(x,u) = 1 + uf(x,u) + \frac{x}{u}\left(f(x,u) - f(x,0)\right)$$

The kernel method solves: $f(x,0) = \dfrac{1-\sqrt{1-4x}}{2x}$.

Enumerating Av(321) via the Av(21)-Machine

When considering the Av(21)-machine, we have to allow bypasses.

Enumerating Av(321) via the Av(21)-Machine

When considering the Av(21)-machine, we have to allow bypasses.

If we're not careful, we'll overcount: for example, the permutation 1 could be generated by

Enumerating Av(321) via the Av(21)-Machine

When considering the Av(21)-machine, we have to allow bypasses.

If we're not careful, we'll overcount: for example, the permutation 1 could be generated by

- a push then a pop, or

Enumerating Av(321) via the Av(21)-Machine

When considering the Av(21)-machine, we have to allow bypasses.

If we're not careful, we'll overcount: for example, the permutation 1 could be generated by

- a push then a pop, or
- a bypass.

Enumerating Av(321) via the Av(21)-Machine

When considering the Av(21)-machine, we have to allow bypasses.

If we're not careful, we'll overcount: for example, the permutation 1 could be generated by

- a push then a pop, or
- a bypass.

To ensure uniqueness of generation sequences, we require that

Enumerating Av(321) via the Av(21)-Machine

When considering the Av(21)-machine, we have to allow bypasses.

If we're not careful, we'll overcount: for example, the permutation 1 could be generated by

- a push then a pop, or
- a bypass.

To ensure uniqueness of generation sequences, we require that

- a bypass is always preferred when possible,

Enumerating Av(321) via the Av(21)-Machine

When considering the Av(21)-machine, we have to allow bypasses.

If we're not careful, we'll overcount: for example, the permutation 1 could be generated by

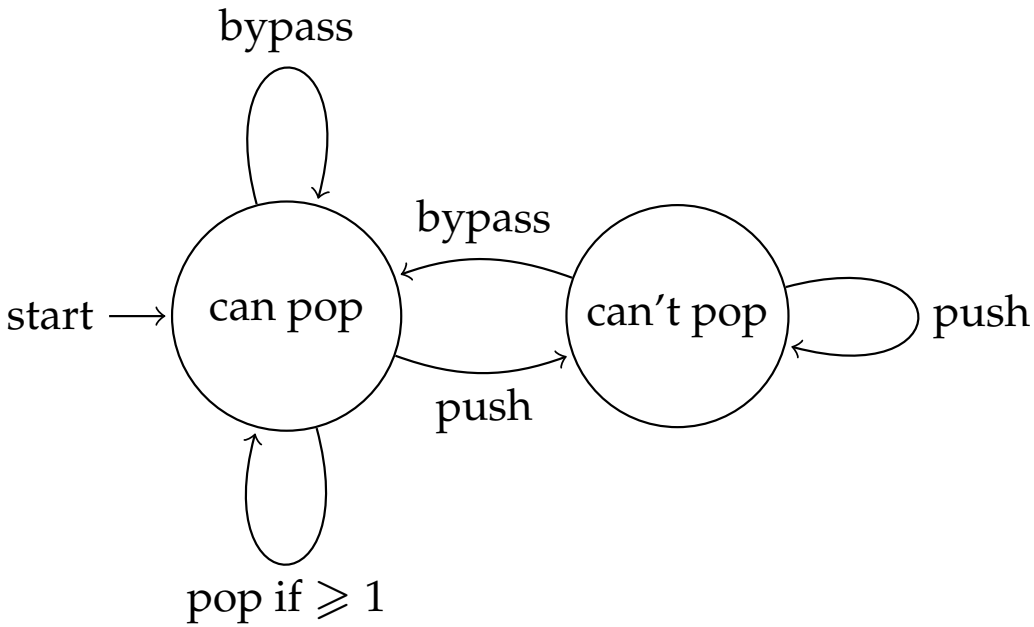
- a push then a pop, or
- a bypass.

To ensure uniqueness of generation sequences, we require that

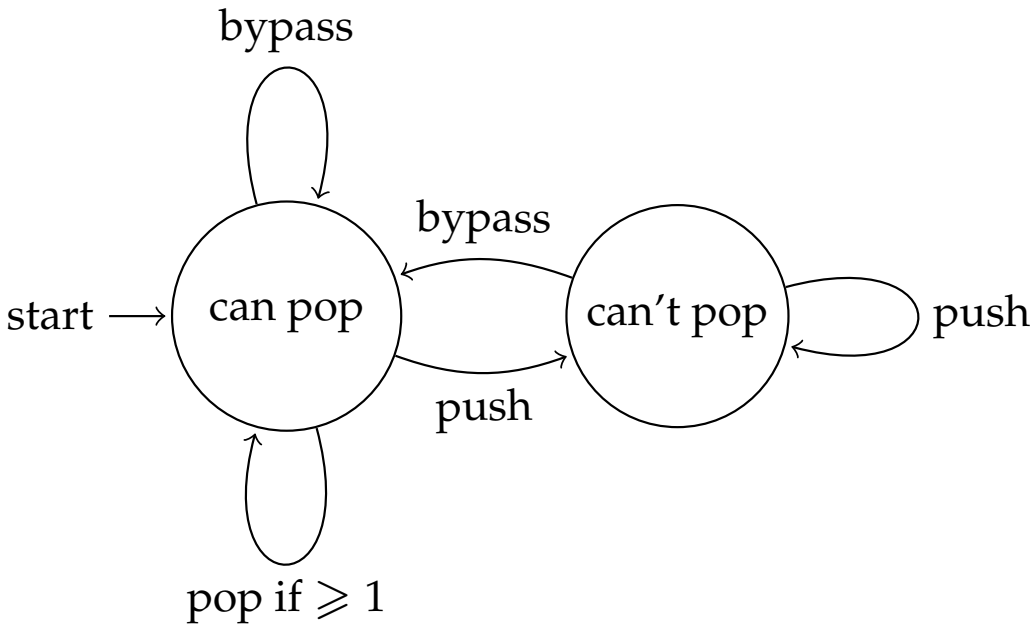
- a bypass is always preferred when possible,
- an entry should be popped as soon as possible.

Equivalently: once you push an entry, you cannot pop any entries until you have bypassed an entry.

Enumerating Av(321) via the Av(21)-Machine



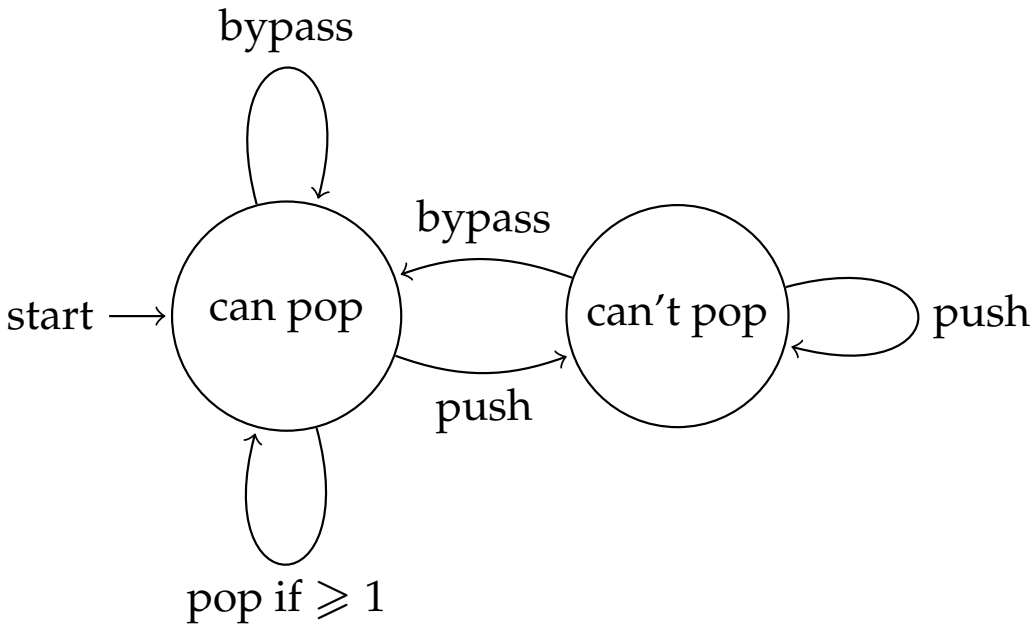
Enumerating Av(321) via the Av(21)-Machine



$$[\text{can pop}]: \quad f(x,u) = 1 + x(f(x,u) + g(x,u)) + \frac{x}{u}(f(x,u) - f(x,0)),$$

$$[\text{can't pop}]: \quad g(x,u) = u(f(x,u) + g(x,u)).$$

Enumerating Av(321) via the Av(21)-Machine



$$[\text{can pop}]: \quad f(x,u) = 1 + x(f(x,u) + g(x,u)) + \frac{x}{u}(f(x,u) - f(x,0)),$$

$$[\text{can't pop}]: \quad g(x,u) = u(f(x,u) + g(x,u)).$$

Solve $g(x,u)$ in terms of $f(x,u) \to$ substitute into $f(x,u) \to$ kernel method $\to$ same answer!

Bijection

The Av(12)-Machine and Av(21)-Machine share the following property:
There is always exactly one location where a new entry can be pushed.

Bijection

The $\mathrm{Av}(12)$-Machine and $\mathrm{Av}(21)$-Machine share the following property:
There is always exactly one location where a new entry can be pushed.

This gives a bijection from $\mathrm{Av}(312)$ to $\mathrm{Av}(321)$. If you use operation sequence S on the $\mathrm{Av}(12)$-Machine to get $\pi \in \mathrm{Av}(312)$, then use the same operation sequence on the $\mathrm{Av}(21)$-Machine to get $f(\pi)$.

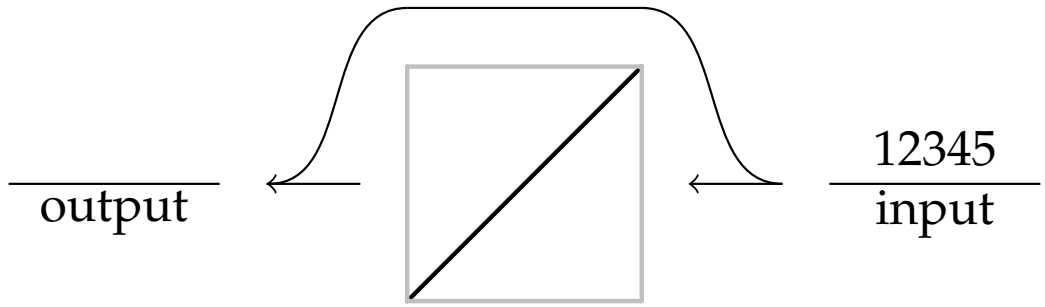
Bijection

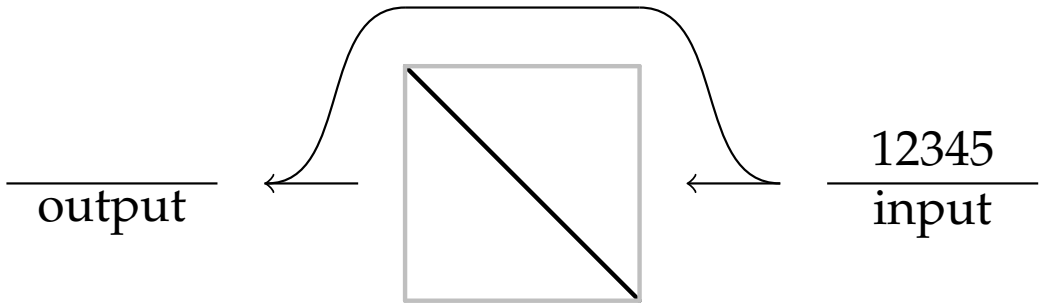
The Av(12)-Machine and Av(21)-Machine share the following property:
There is always exactly one location where a new entry can be pushed.

This gives a bijection from Av(312) to Av(321). If you use operation sequence S on the Av(12)-Machine to get $\pi \in \mathrm{Av}(312)$, then use the same operation sequence on the Av(21)-Machine to get $f(\pi)$.

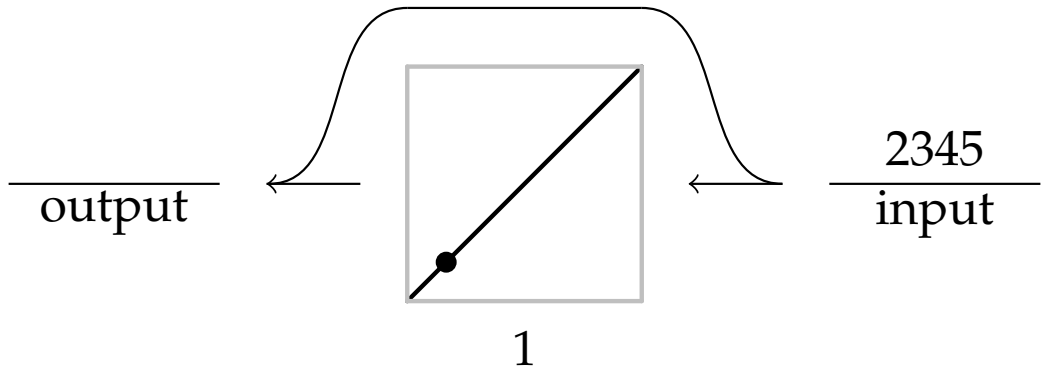
(To make this easier to describe, we tack the bypass back onto the Av(12)-Machine, even though it makes no difference.)

Bijection

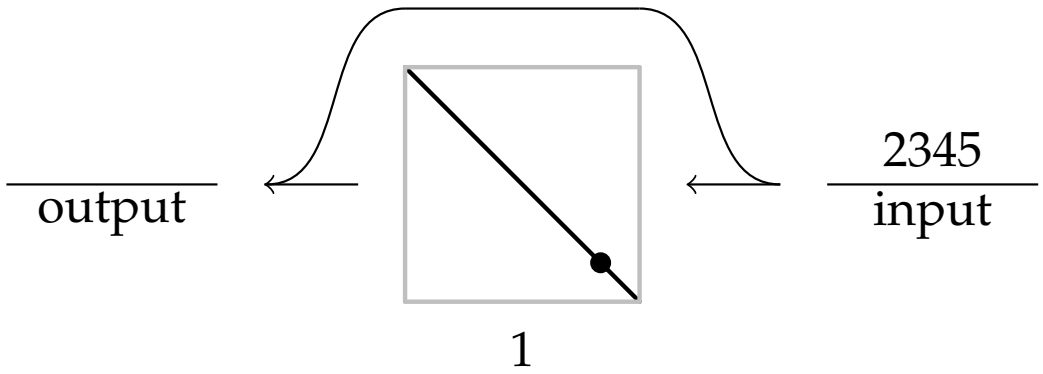




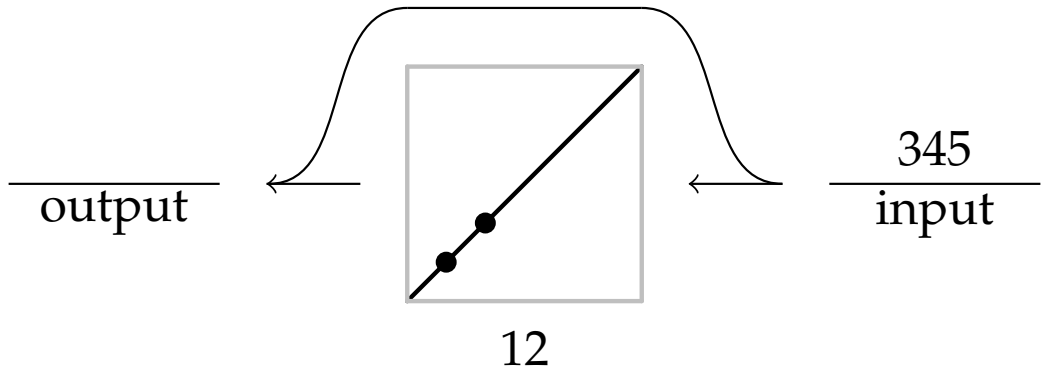
Bijection



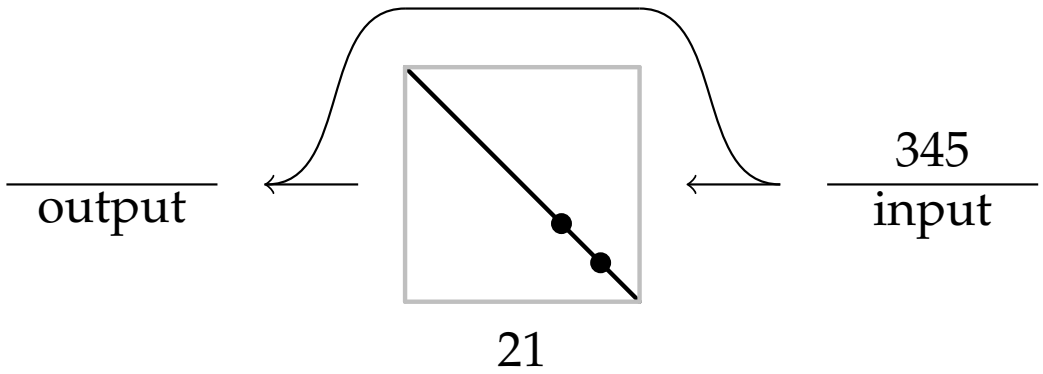
push



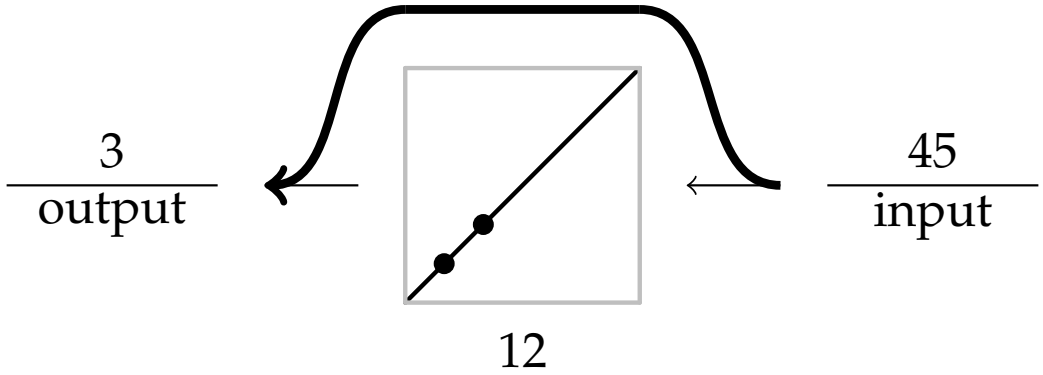
Bijection



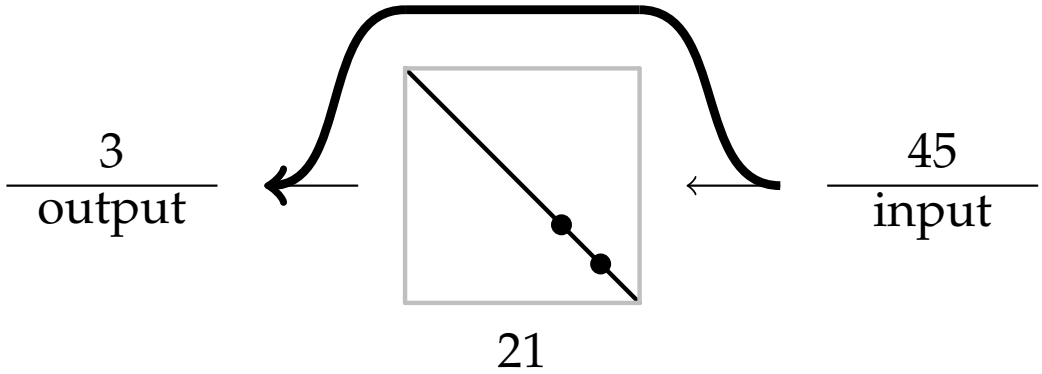
push



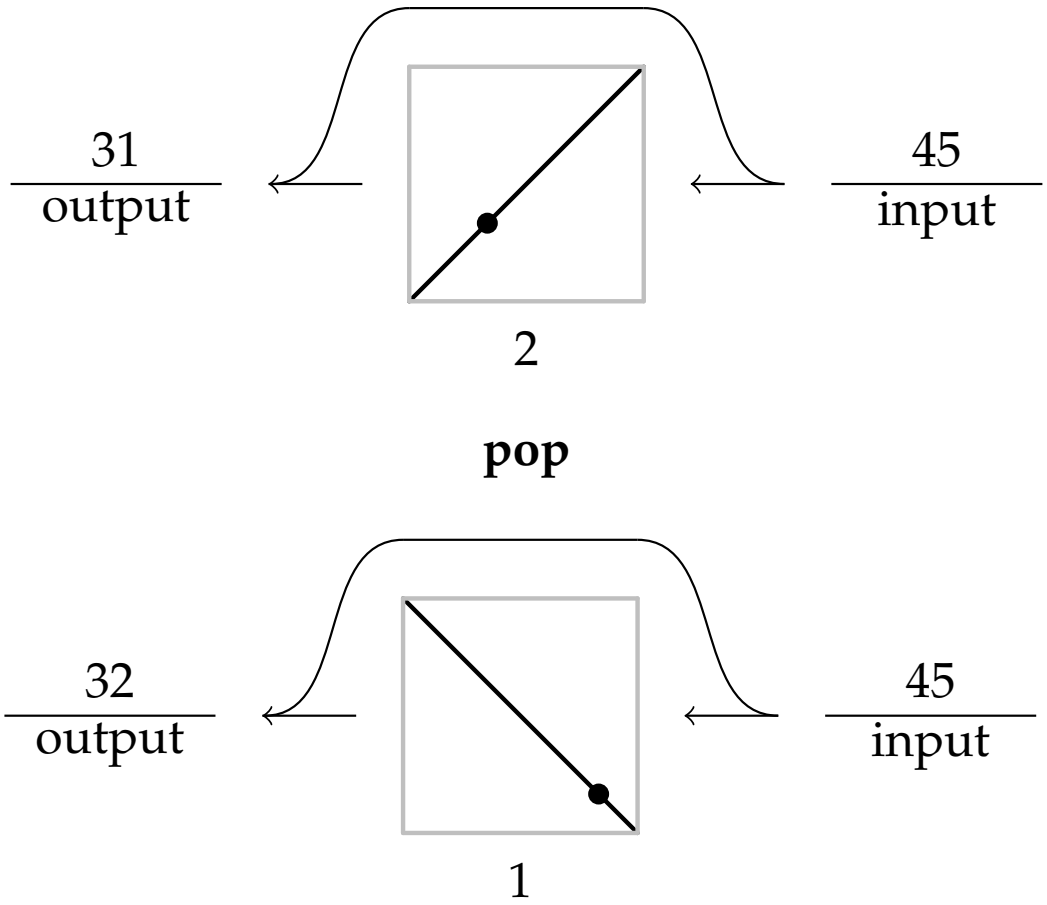
Bijection



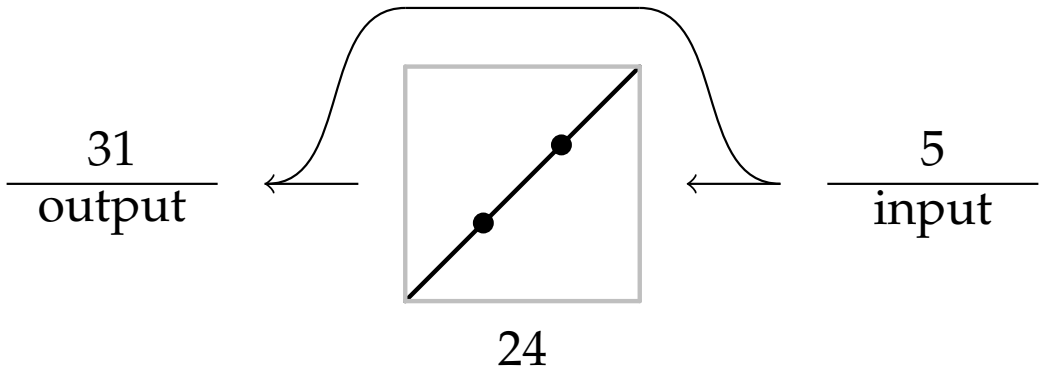
bypass



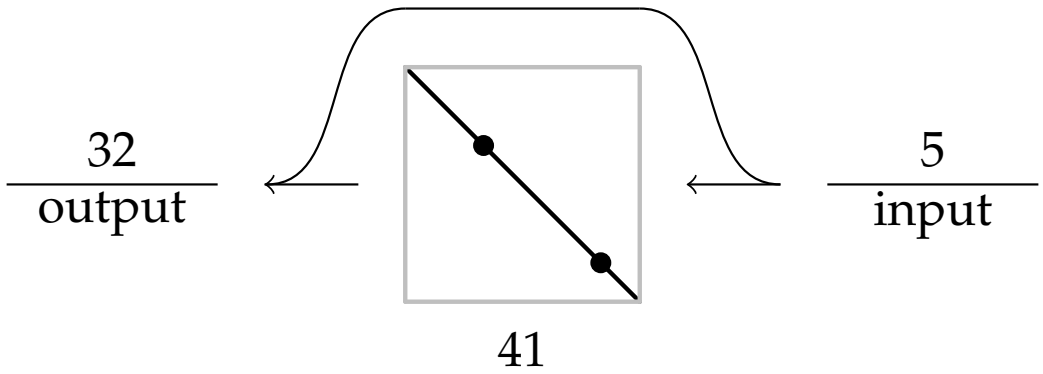
Bijection



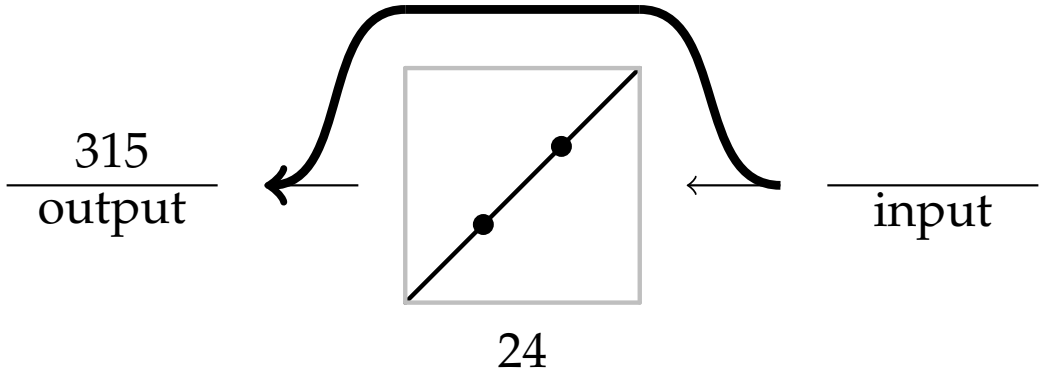
Bijection



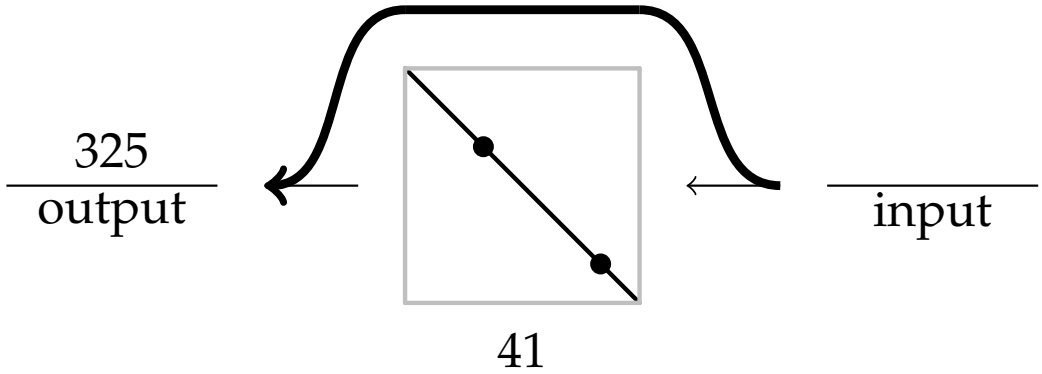
push



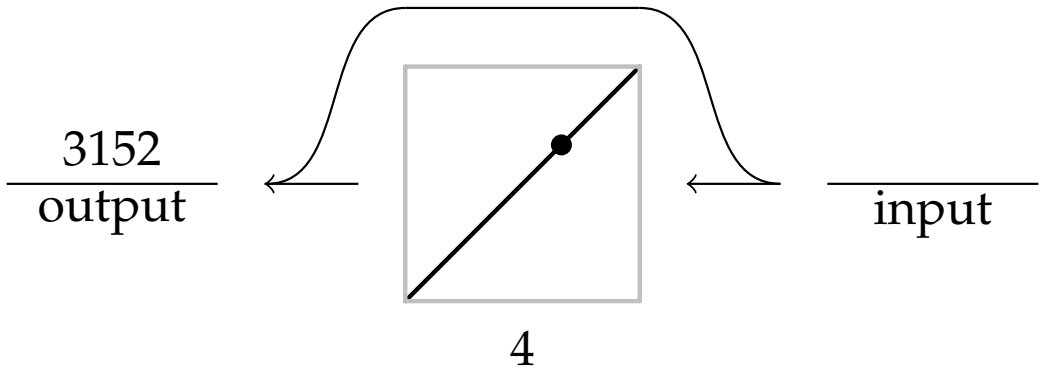
Bijection



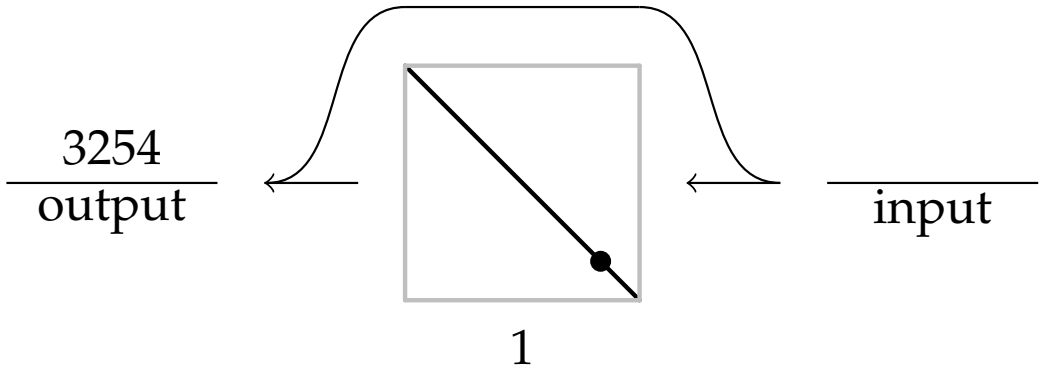
bypass



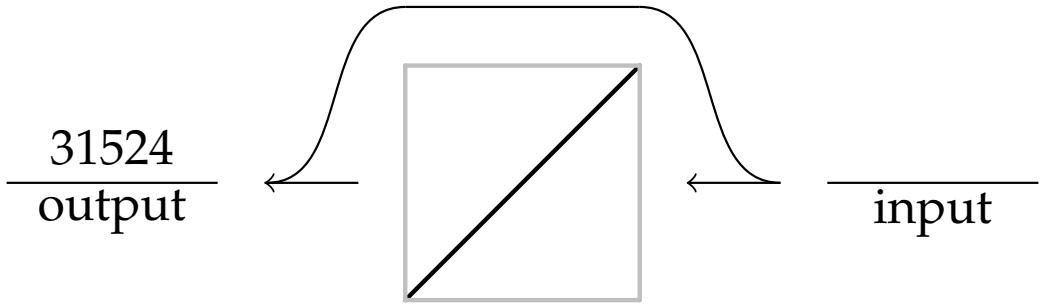
Bijection



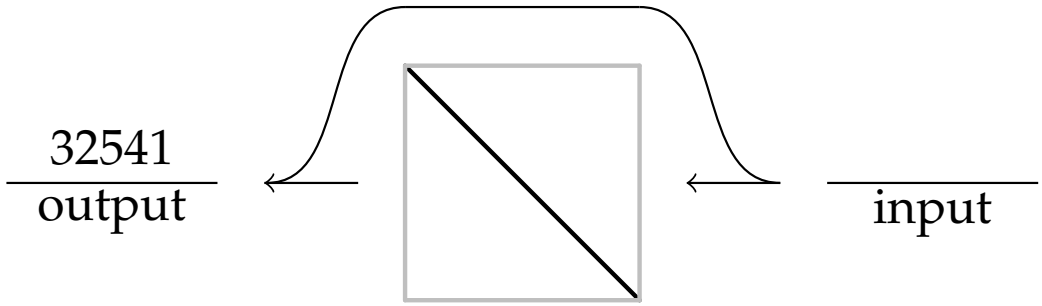
pop



Bijection



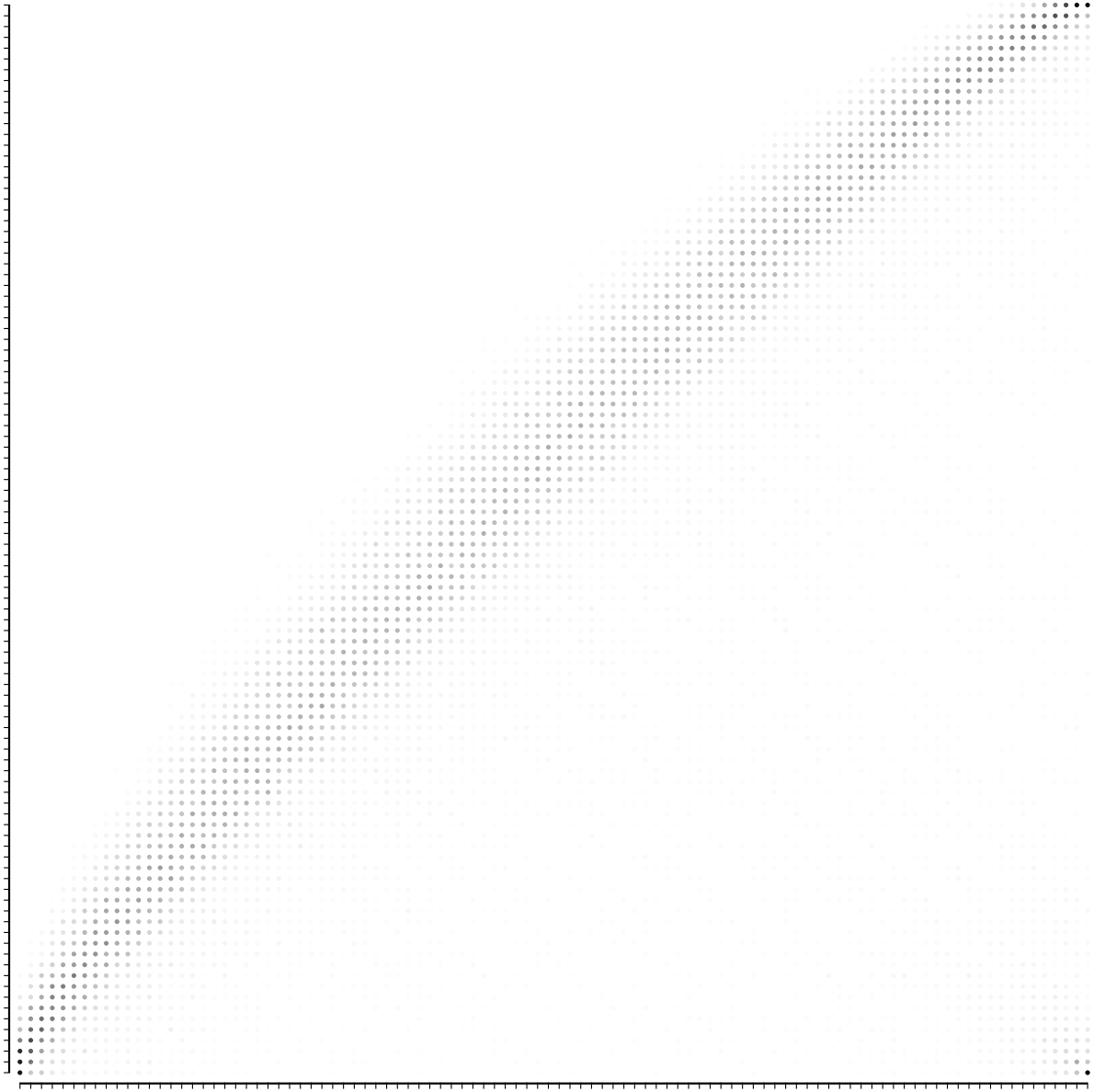
pop



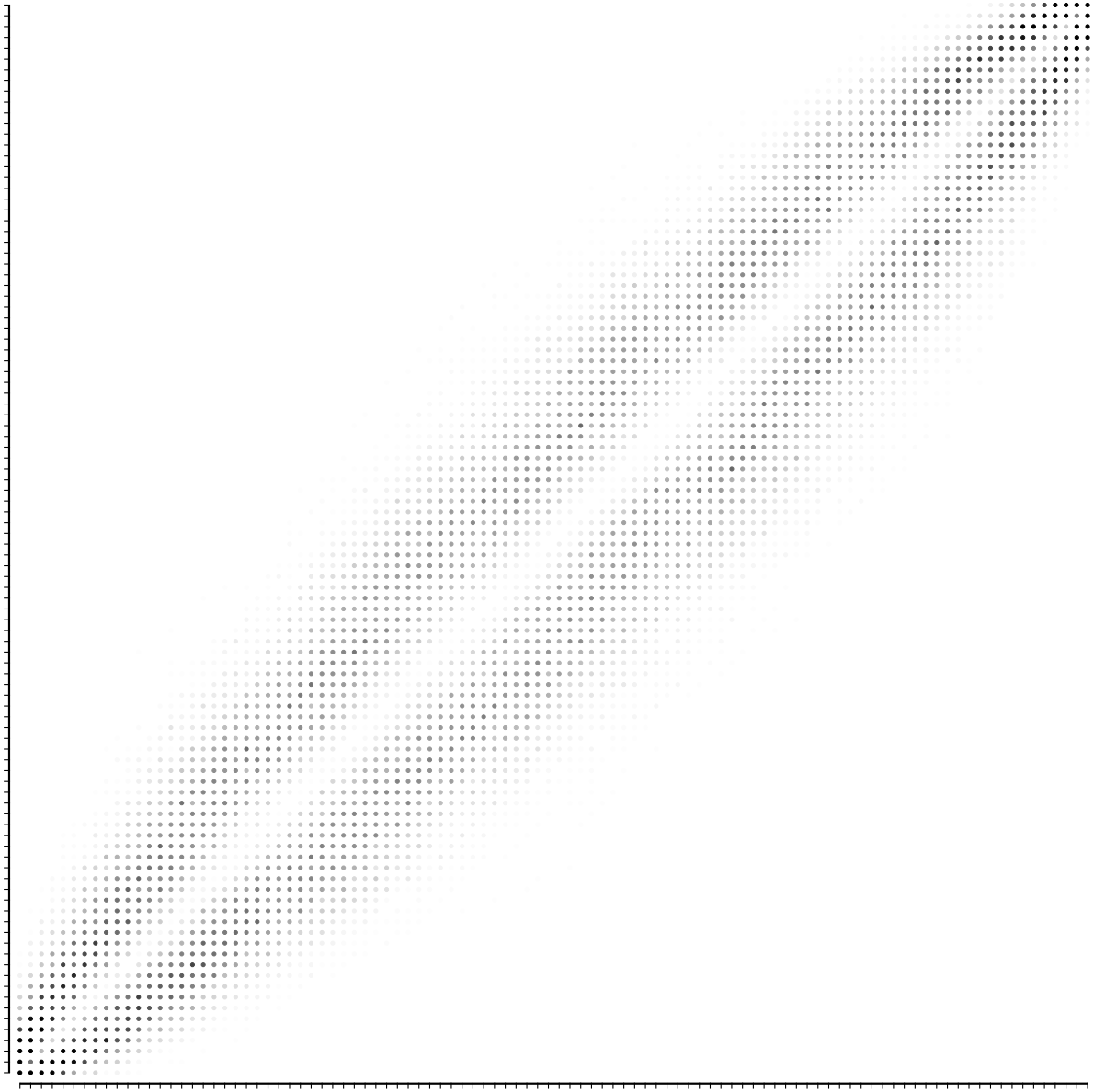
Random Sampling

Any combinatorial family that can be counted with *dynamic programming* can be sampled from uniformly at random.

Random Sampling



Random Sampling



Enumerating the Schröder Classes

A *2×4 class* is a permutation class with two basis elements of length 4.

Enumerating the Schröder Classes

A *2×4 class* is a permutation class with two basis elements of length 4.

- *Example:* $\mathrm{Av}(1324, 2413)$ is a 2×4 class.

Enumerating the Schröder Classes

A *2×4 class* is a permutation class with two basis elements of length 4.

- *Example:* $\mathrm{Av}(1324, 2413)$ is a 2×4 class.

Up to symmetries of the permutation containment order there are 56 essentially different 2×4 classes.

Enumerating the Schröder Classes

A *2×4 class* is a permutation class with two basis elements of length 4.

- *Example:* $\mathrm{Av}(1324, 2413)$ is a 2×4 class.

Up to symmetries of the permutation containment order there are 56 essentially different 2×4 classes.

Some of these 56 2×4 classes have the same enumeration despite being non-isomorphic. There are precisely 38 different enumerations for the 2×4 classes.

Enumerating the Schröder Classes

A *2×4 class* is a permutation class with two basis elements of length 4.

- *Example:* $\mathrm{Av}(1324, 2413)$ is a 2×4 class.

Up to symmetries of the permutation containment order there are 56 essentially different 2×4 classes.

Some of these 56 2×4 classes have the same enumeration despite being non-isomorphic. There are precisely 38 different enumerations for the 2×4 classes.

Of these 38 classes, the exact enumerations are known for 35 of them. (More on this later!)

Enumerating the Schröder Classes

There are ten 2×4 classes that are enumerated by the Schröder numbers: $1, 2, 6, 22, 90, \ldots$.

Enumerating the Schröder Classes

There are ten 2×4 classes that are enumerated by the Schröder numbers: $1, 2, 6, 22, 90, \ldots$.

Six of them are of the form $\mathrm{Av}({}^{+}\beta_1, {}^{+}\beta_2)$ and so they are generated by $\mathcal{C}$-machines.

Enumerating the Schröder Classes

There are ten 2×4 classes that are enumerated by the Schröder numbers: $1, 2, 6, 22, 90, \ldots$.

Six of them are of the form $\mathrm{Av}({}^{+}\beta_1, {}^{+}\beta_2)$ and so they are generated by $\mathcal{C}$-machines.

- *Example:* $\mathrm{Av}(4231, 4132)$ is enumerated by the Schröder numbers, and is generated by the $\mathrm{Av}(231, 132)$ machine.

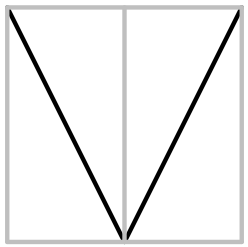
Enumerating the Schröder Classes

There are ten 2×4 classes that are enumerated by the Schröder numbers: $1, 2, 6, 22, 90, \ldots$.

Six of them are of the form $\mathrm{Av}({}^{+}\beta_1, {}^{+}\beta_2)$ and so they are generated by $\mathcal{C}$-machines.

- *Example:* $\mathrm{Av}(4231, 4132)$ is enumerated by the Schröder numbers, and is generated by the $\mathrm{Av}(231, 132)$ machine.

Example: $\mathrm{Av}(231, 132)$.



Enumerating the Schröder Classes

A small adjustment to the earlier functional equations gives:

$$f(x,u) = 1 + x(f(x,u) + g(x,u)) + \frac{x}{u}(f(x,u) - f(x,0)),$$
$$g(x,u) = 2u((f(x,u) - f(x,0)) + g(x,u)) + uf(x,0).$$

Enumerating the Schröder Classes

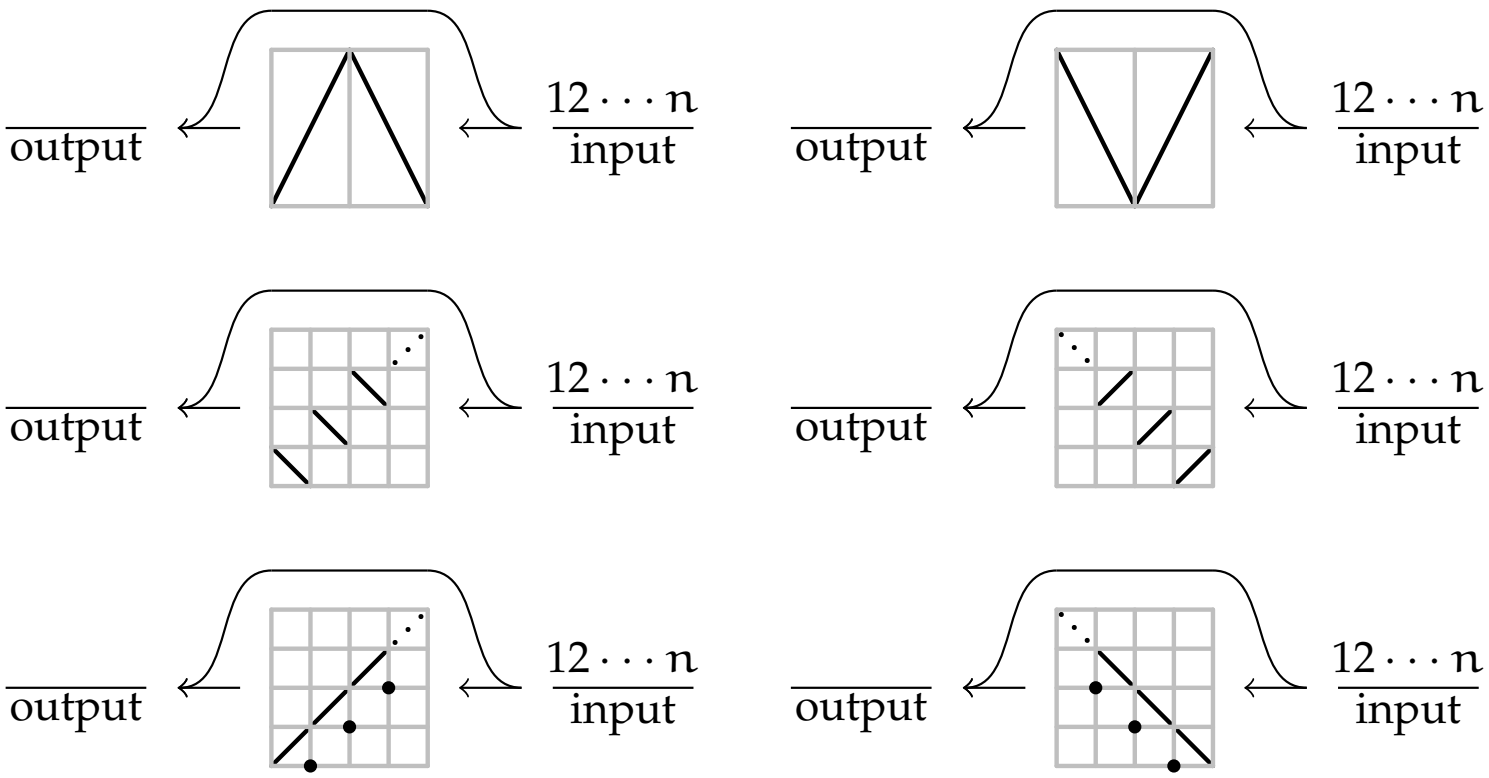
A small adjustment to the earlier functional equations gives:

$$f(x,u) = 1 + x(f(x,u) + g(x,u)) + \frac{x}{u}(f(x,u) - f(x,0)),$$
$$g(x,u) = 2u((f(x,u) - f(x,0)) + g(x,u)) + uf(x,0).$$

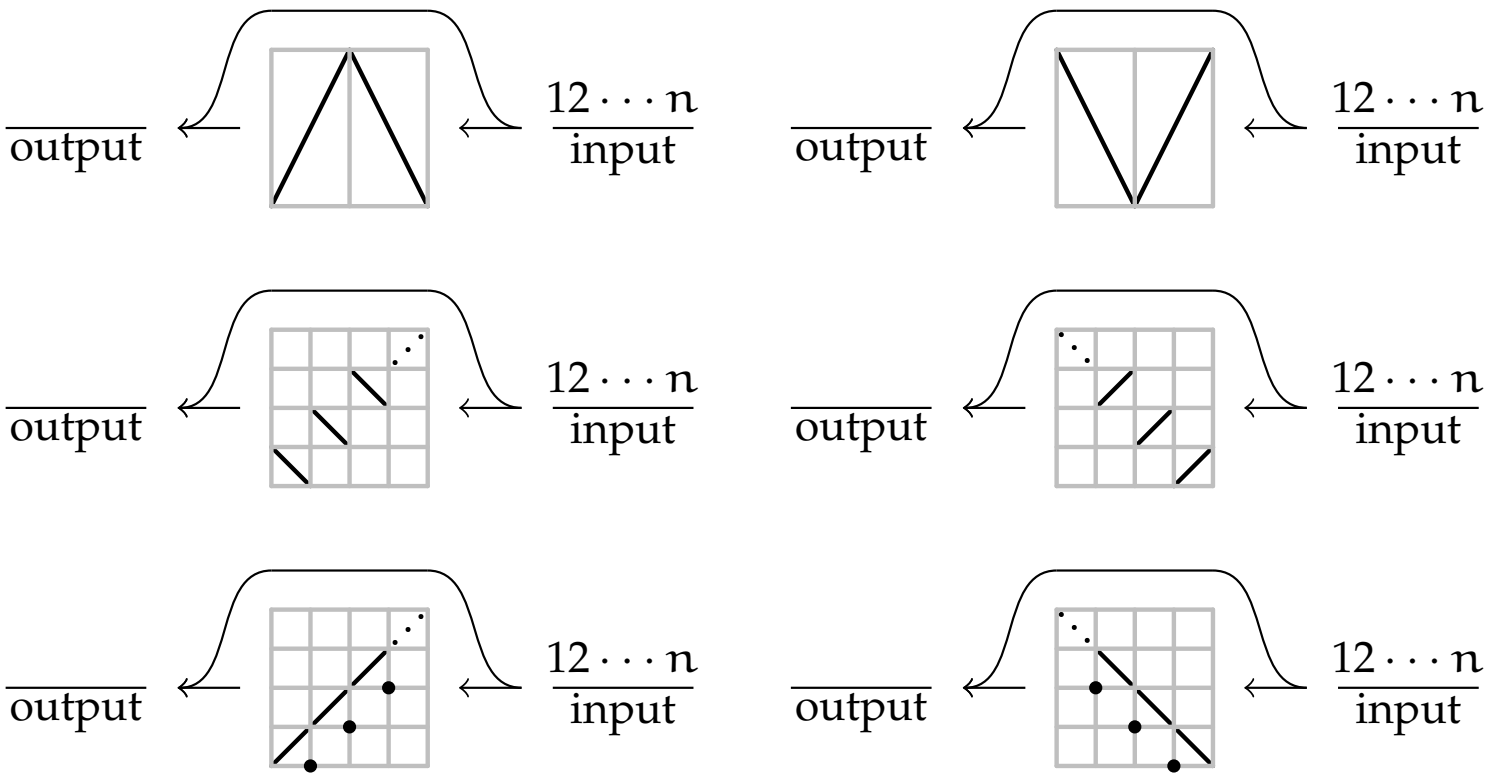
An application of the kernel method gives the generating function for the Schröder numbers.

$$f(x,0) = \frac{3 - x - \sqrt{1 - 6x + x^2}}{2}$$

Enumerating the Schröder Classes

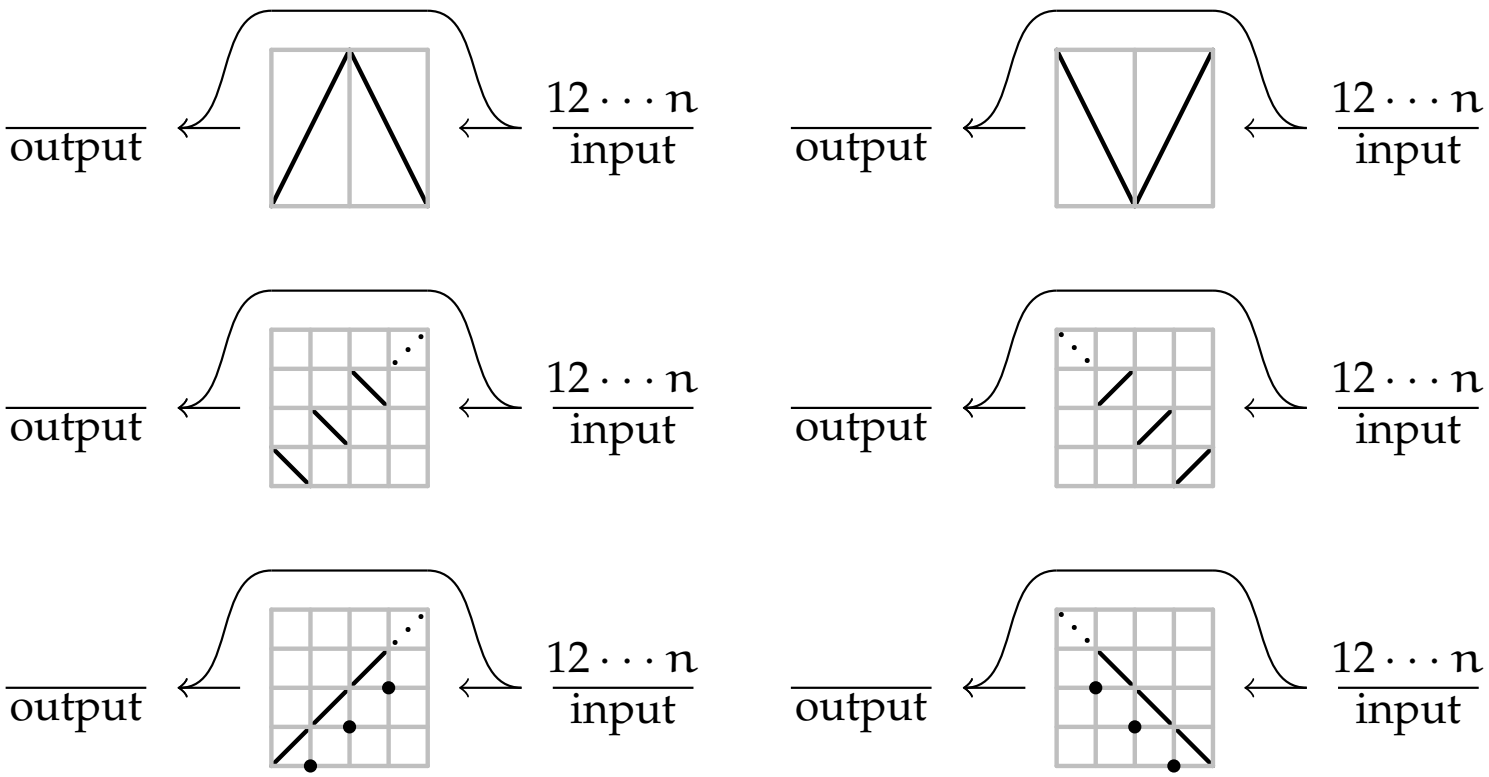


Enumerating the Schröder Classes



All are enumerated by the Schröder numbers.

Enumerating the Schröder Classes



All are enumerated by the Schröder numbers.

Free bijections between every pair of classes.

2×4 Classes

Enumerations are known for all but three:

- $\mathrm{Av}(4123, 4231)$
- $\mathrm{Av}(4123, 4312)$
- $\mathrm{Av}(4231, 4321)$

2×4 Classes

Enumerations are known for all but three:

- $\mathrm{Av}(4123, 4231)$
- $\mathrm{Av}(4123, 4312)$
- $\mathrm{Av}(4231, 4321)$

These three can all be modeled with $\mathcal{C}$-machines.

$\mathrm{Av}(4123, 4231)$

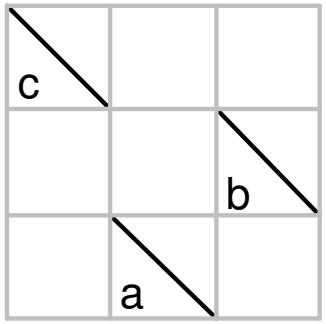
$\mathrm{Av}(4123, 4231)$ is generated by the $\mathrm{Av}(123, 231)$-machine.

The class $\mathrm{Av}(123, 231)$ is a geometric grid class.

$\mathrm{Av}(4123, 4231)$

$\mathrm{Av}(4123, 4231)$ is generated by the $\mathrm{Av}(123, 231)$-machine.

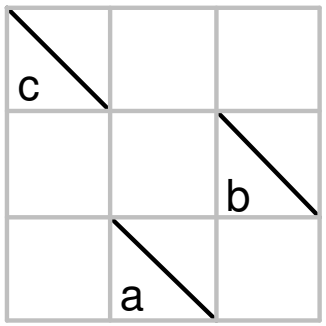
The class $\mathrm{Av}(123, 231)$ is a geometric grid class.



$\mathrm{Av}(4123, 4231)$

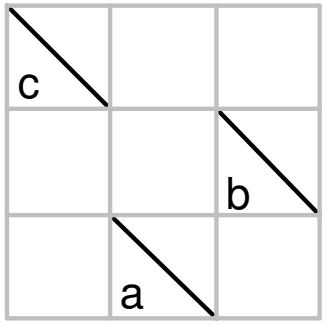
$\mathrm{Av}(4123, 4231)$ is generated by the $\mathrm{Av}(123, 231)$-machine.

The class $\mathrm{Av}(123, 231)$ is a geometric grid class.



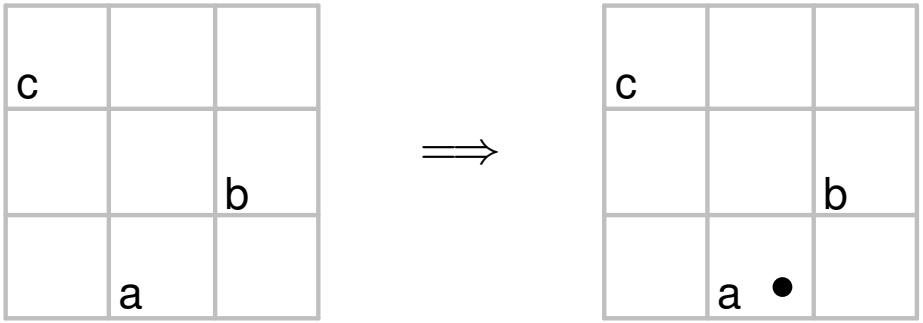
The states of the machine can be represented by 4-tuples (a, b, c, P) where a, b, and c are the number of a, b, c entries, and P is a boolean representing whether we can or can't pop.

$\mathrm{Av}(4123, 4231)$



We can describe the state transitions:

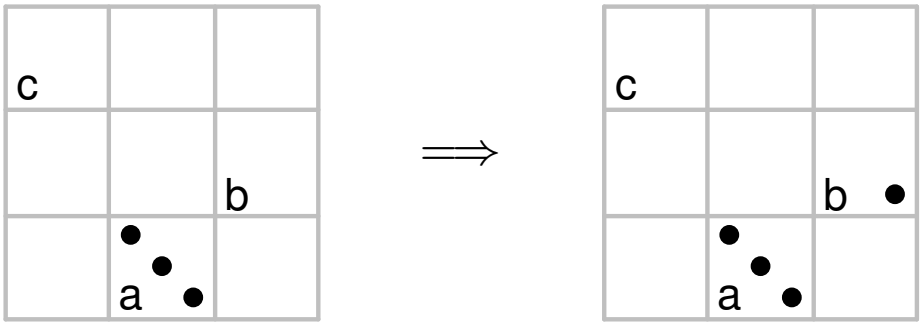
Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,\mathsf{T}) \longrightarrow \{(1,0,0,\mathsf{F}),(0,0,0,\mathsf{T})\}$

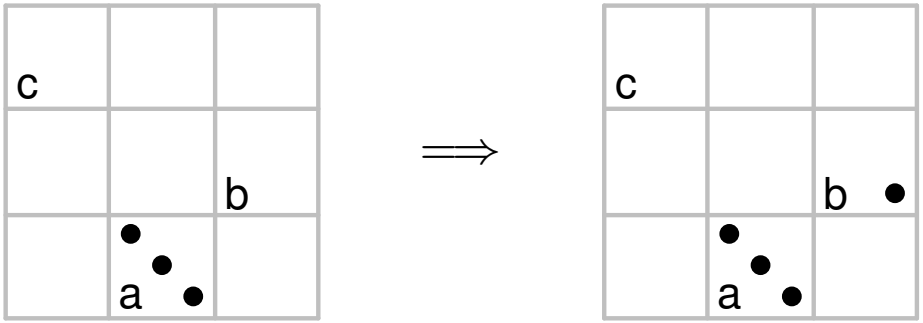
Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,\mathrm{T}) \longrightarrow \{(1,0,0,\mathrm{F}),(0,0,0,\mathrm{T})\}$
- $(a,0,0,\mathrm{F}) \longrightarrow \{(a+1,0,0,\mathrm{F}),(a,1,0,\mathrm{F}),(a,0,0,\mathrm{T})\}$

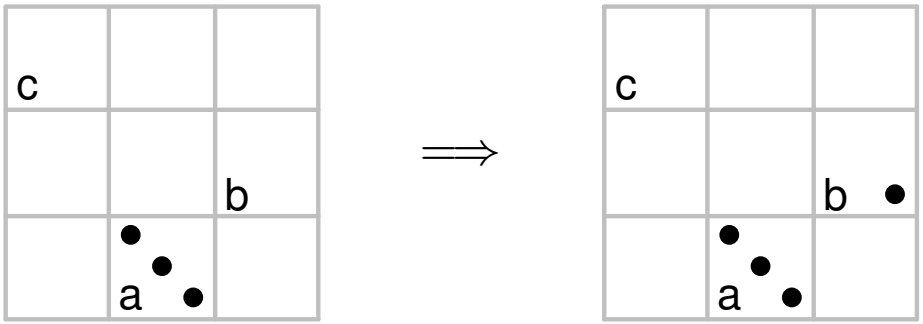
Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,\mathrm{T}) \longrightarrow \{(1,0,0,\mathrm{F}),(0,0,0,\mathrm{T})\}$
- $(a,0,0,\mathrm{F}) \longrightarrow \{(a+1,0,0,\mathrm{F}),(a,1,0,\mathrm{F}),(a,0,0,\mathrm{T})\}$
- $(a,0,0,\mathrm{T}) \longrightarrow \{(a+1,0,0,\mathrm{F}),(a,1,0,\mathrm{F}),(a,0,0,\mathrm{T}),(a-1,0,0,\mathrm{T})\}$

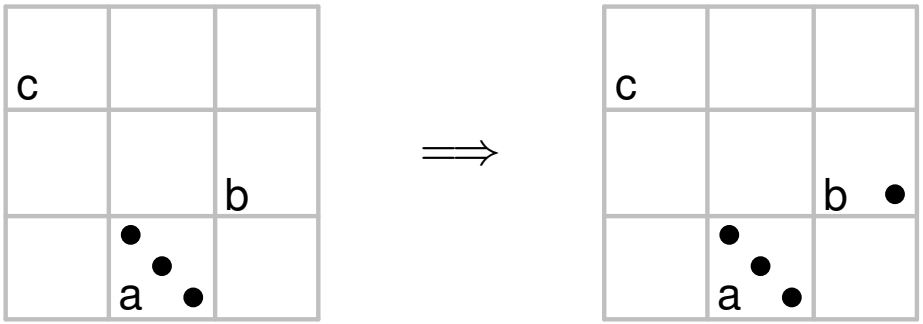
Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,\mathrm{T}) \longrightarrow \{(1,0,0,\mathrm{F}),(0,0,0,\mathrm{T})\}$
- $(a,0,0,\mathrm{F}) \longrightarrow \{(a+1,0,0,\mathrm{F}),(a,1,0,\mathrm{F}),(a,0,0,\mathrm{T})\}$
- $(a,0,0,\mathrm{T}) \longrightarrow \{(a+1,0,0,\mathrm{F}),(a,1,0,\mathrm{F}),(a,0,0,\mathrm{T}),(a-1,0,0,\mathrm{T})\}$
- $(a,b,0,\mathrm{F}) \longrightarrow \{(a,b+1,0,\mathrm{F}),(a,b,1,\mathrm{F}),(a,b,0,\mathrm{T})\}$

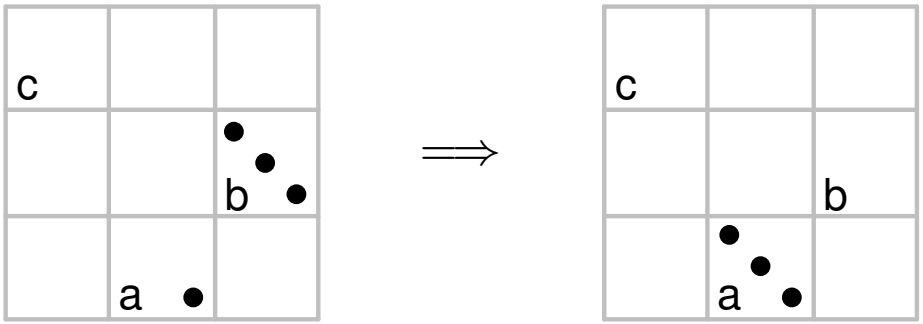
Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,T) \longrightarrow \{(1,0,0,F),(0,0,0,T)\}$
- $(a,0,0,F) \longrightarrow \{(a+1,0,0,F),(a,1,0,F),(a,0,0,T)\}$
- $(a,0,0,T) \longrightarrow \{(a+1,0,0,F),(a,1,0,F),(a,0,0,T),(a-1,0,0,T)\}$
- $(a,b,0,F) \longrightarrow \{(a,b+1,0,F),(a,b,1,F),(a,b,0,T)\}$
- $(a,b,0,T) \longrightarrow \{(a,b+1,0,F),(a,b,1,F),(a,b,0,T),(a-1,b,0,T)\}$, $(a \geqslant 2)$

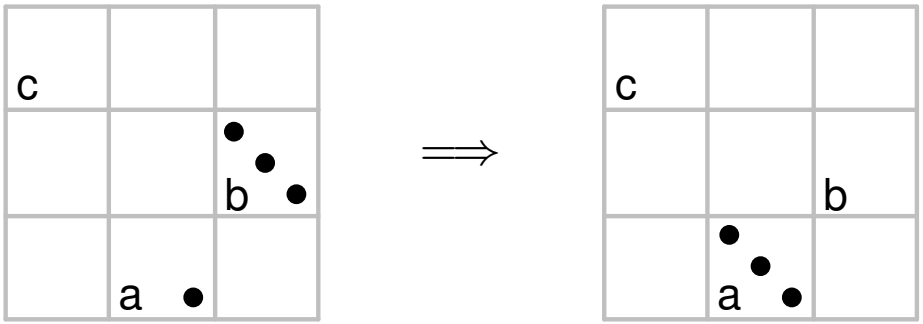
Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,\mathrm{T}) \longrightarrow \{(1,0,0,\mathrm{F}),(0,0,0,\mathrm{T})\}$
- $(a,0,0,\mathrm{F}) \longrightarrow \{(a+1,0,0,\mathrm{F}),(a,1,0,\mathrm{F}),(a,0,0,\mathrm{T})\}$
- $(a,0,0,\mathrm{T}) \longrightarrow \{(a+1,0,0,\mathrm{F}),(a,1,0,\mathrm{F}),(a,0,0,\mathrm{T}),(a-1,0,0,\mathrm{T})\}$
- $(a,b,0,\mathrm{F}) \longrightarrow \{(a,b+1,0,\mathrm{F}),(a,b,1,\mathrm{F}),(a,b,0,\mathrm{T})\}$
- $(a,b,0,\mathrm{T}) \longrightarrow \{(a,b+1,0,\mathrm{F}),(a,b,1,\mathrm{F}),(a,b,0,\mathrm{T}),(a-1,b,0,\mathrm{T})\}$, $(a \geqslant 2)$
- $(1,b,0,\mathrm{T}) \longrightarrow \{(1,b+1,0,\mathrm{F}),(1,b,1,\mathrm{F}),(1,b,0,\mathrm{T}),(b,0,0,\mathrm{T})\}$

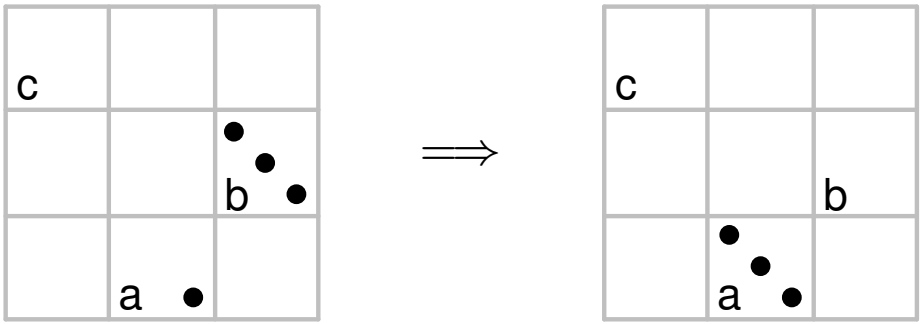
Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,T) \longrightarrow \{(1,0,0,F),(0,0,0,T)\}$
- $(a,0,0,F) \longrightarrow \{(a+1,0,0,F),(a,1,0,F),(a,0,0,T)\}$
- $(a,0,0,T) \longrightarrow \{(a+1,0,0,F),(a,1,0,F),(a,0,0,T),(a-1,0,0,T)\}$
- $(a,b,0,F) \longrightarrow \{(a,b+1,0,F),(a,b,1,F),(a,b,0,T)\}$
- $(a,b,0,T) \longrightarrow \{(a,b+1,0,F),(a,b,1,F),(a,b,0,T),(a-1,b,0,T)\}$, $(a \geqslant 2)$
- $(1,b,0,T) \longrightarrow \{(1,b+1,0,F),(1,b,1,F),(1,b,0,T),(b,0,0,T)\}$
- $(a,b,c,F) \longrightarrow \{(a,b,c+1,F),(a,b,c,T)\}$

Av(4123, 4231)



We can describe the state transitions:

- $(0,0,0,T) \longrightarrow \{(1,0,0,F),(0,0,0,T)\}$
- $(a,0,0,F) \longrightarrow \{(a+1,0,0,F),(a,1,0,F),(a,0,0,T)\}$
- $(a,0,0,T) \longrightarrow \{(a+1,0,0,F),(a,1,0,F),(a,0,0,T),(a-1,0,0,T)\}$
- $(a,b,0,F) \longrightarrow \{(a,b+1,0,F),(a,b,1,F),(a,b,0,T)\}$
- $(a,b,0,T) \longrightarrow \{(a,b+1,0,F),(a,b,1,F),(a,b,0,T),(a-1,b,0,T)\}$, $(a \geqslant 2)$
- $(1,b,0,T) \longrightarrow \{(1,b+1,0,F),(1,b,1,F),(1,b,0,T),(b,0,0,T)\}$
- $(a,b,c,F) \longrightarrow \{(a,b,c+1,F),(a,b,c,T)\}$
- $(a,b,c,T) \longrightarrow \{(a,b,c+1,F),(a,b,c,T),(a,b,c-1,T)\}$

The Av(123, 231)-Machine

With a little work, we can translate the state transitions into a set of functional equations:

$$A(a,x) = 1 + \frac{x}{a}(A(a,x) - A(0,x)) + aA(a,x) + xB(0,a,x),$$

$$B(a,b,x) = \frac{1}{a}(A(a,x) - A(0,x))\frac{bC}{1-b} + B(a,b,x)\frac{bC}{1-b} + \frac{x}{a}(1+C)(B(a,b,x) - B(0,b,x)).$$

The Av(123, 231)-Machine

With a little work, we can translate the state transitions into a set of functional equations:

$$A(a,x) = 1 + \frac{x}{a}(A(a,x) - A(0,x)) + aA(a,x) + xB(0,a,x),$$

$$B(a,b,x) = \frac{1}{a}(A(a,x) - A(0,x))\frac{bC}{1-b} + B(a,b,x)\frac{bC}{1-b} + \frac{x}{a}(1+C)(B(a,b,x) - B(0,b,x)).$$

The generating function for the class is $A(0,x)$, but we have no idea how to solve these equations.

The Av(123, 231)-Machine

Using the state transitions, dynamic programming, and Amazon cloud computing, we have computed the first 1000 terms of Av(4123, 4231).

The Av(123, 231)-Machine

Using the state transitions, dynamic programming, and Amazon cloud computing, we have computed the first 1000 terms of Av(4123, 4231).

We've written software that takes initial terms of a sequence and tries to produce a conjecture for the actual generating function. It searches the space of rational, algebraic, D-finite, and D-algebraic functions.

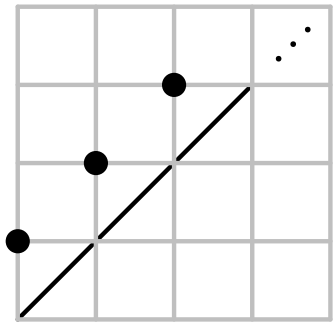
The Av(123, 231)-Machine

Using the state transitions, dynamic programming, and Amazon cloud computing, we have computed the first 1000 terms of Av(4123, 4231).

We've written software that takes initial terms of a sequence and tries to produce a conjecture for the actual generating function. It searches the space of rational, algebraic, D-finite, and D-algebraic functions.

Despite having 1000 initial terms of Av(4123, 4231), we can't guess any generating function!

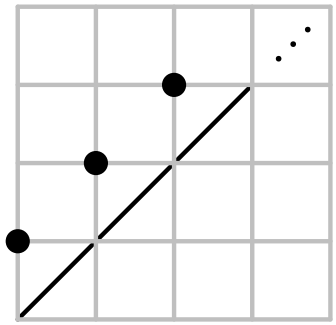
Three More Exotic Machines



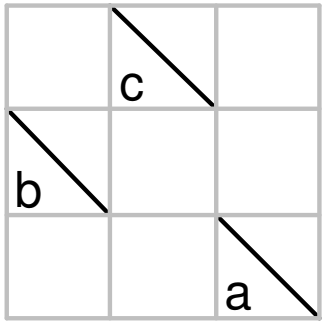
600 terms

$\mathrm{Av}(4231, 4321)$

Three More Exotic Machines

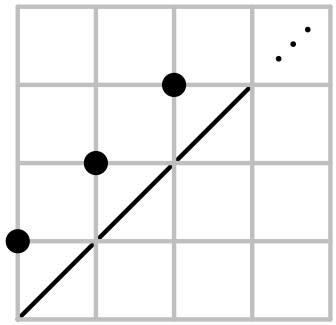


600 terms
$\mathrm{Av}(4231, 4321)$

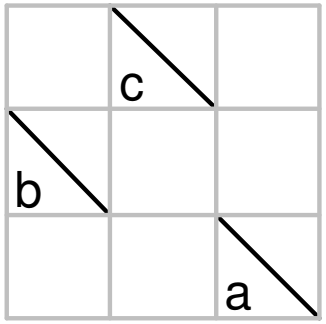


1000 terms
$\mathrm{Av}(4123, 4312)$

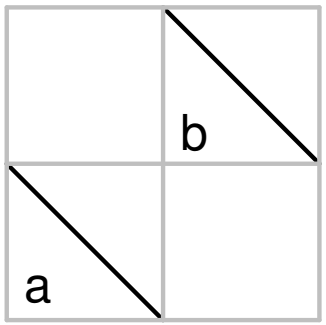
Three More Exotic Machines



600 terms
$\mathrm{Av}(4231, 4321)$



1000 terms
$\mathrm{Av}(4123, 4312)$



5000 terms
$\mathrm{Av}(4123, 4231, 4312)$

Still...

Even though we don't have a generating function, we can still:

Still...

Even though we don't have a generating function, we can still:

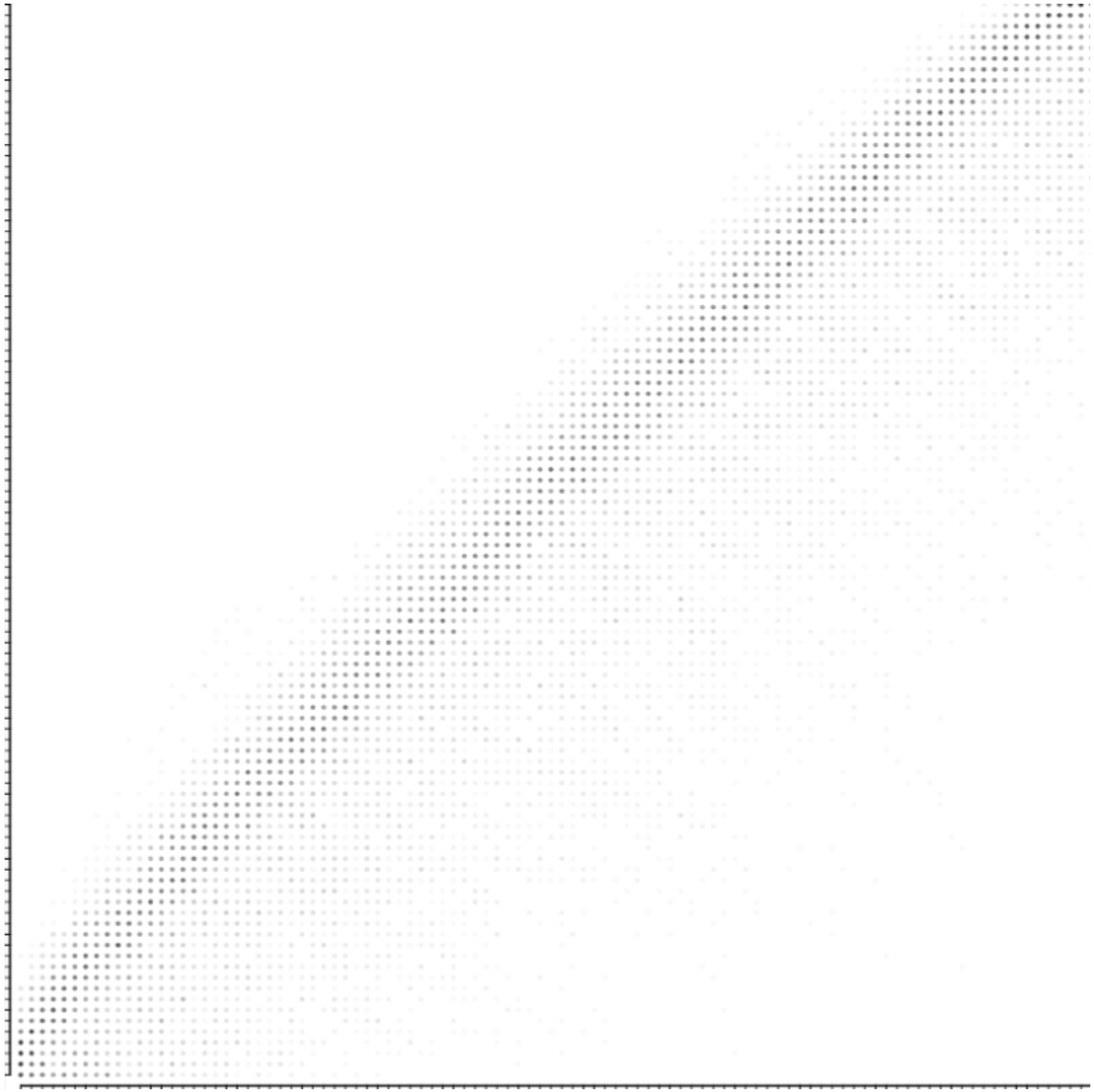
- Sample uniformly at random

Still...

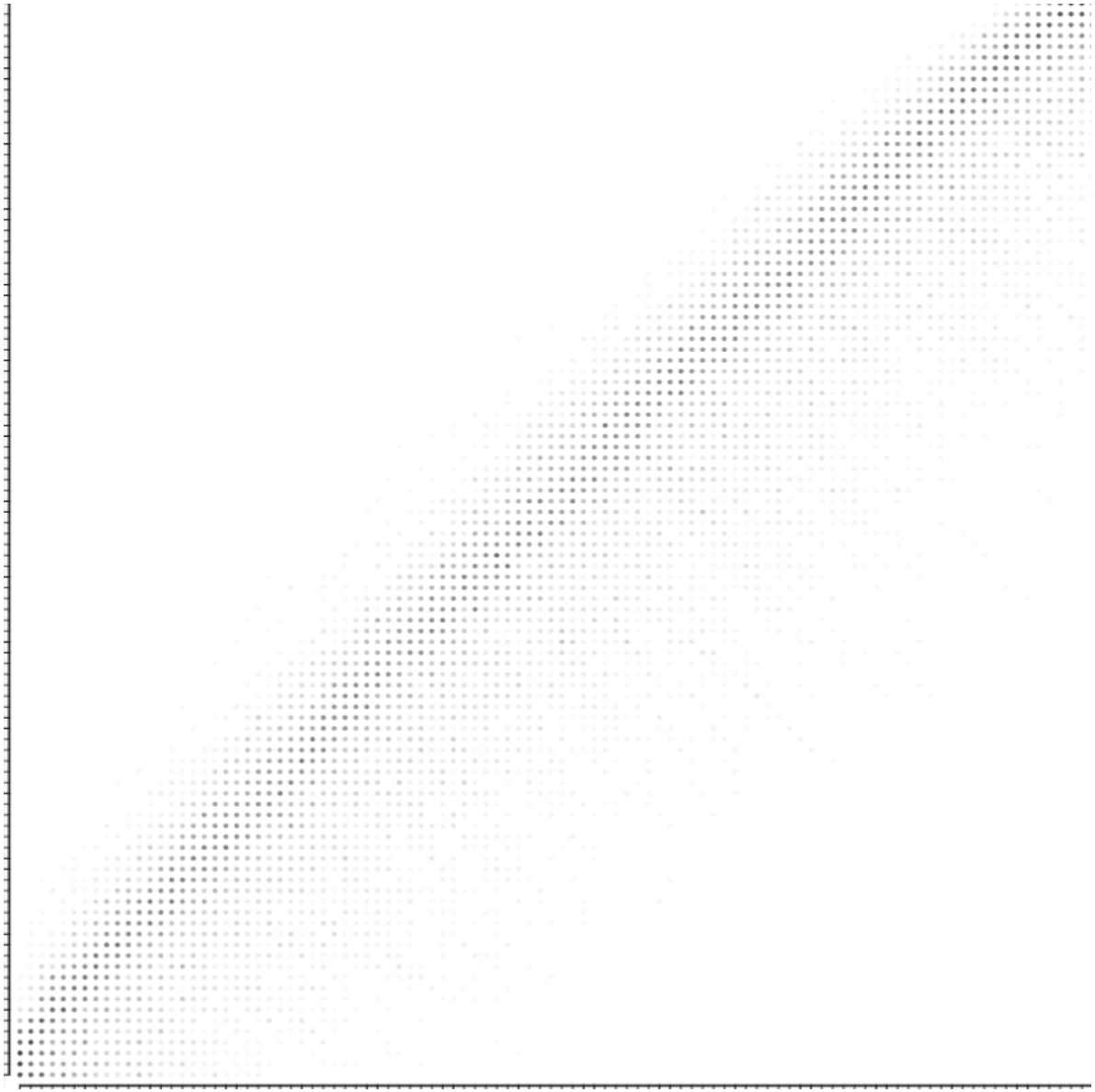
Even though we don't have a generating function, we can still:

- Sample uniformly at random
- Use many initial terms to approximate the singularity structure and asymptotic behavior

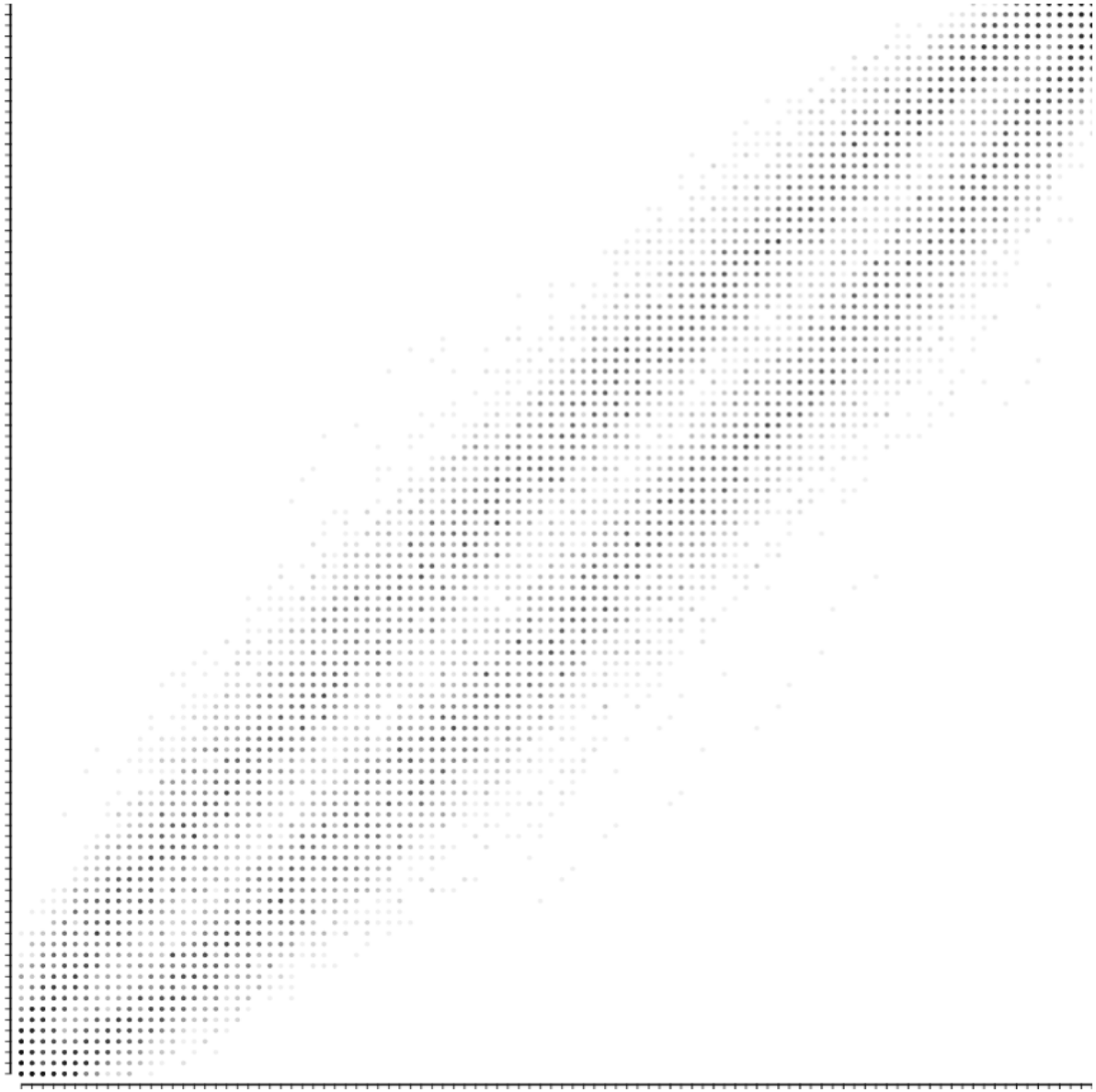
$\mathrm{Av}(4123, 4231)$



$\mathrm{Av}(4123, 4312)$



$\mathrm{Av}(4231, 4321)$



Singularity Structure – Av(4123,4231,4312)

Dominant singularity at

$$x = 0.22379330134707544472559413080653078228254418356103196904827043215739850630915030$$

with critical exponent

$$-1.00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000$$

Singularity Structure – Av(4123,4231,4312)

Dominant singularity at

$$x = 0.22379330134707544472559413080653078228254418356103196904827043215739850630915030$$

with critical exponent

$$-1.00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000$$

Other singularities:

- 0.23806375980802816276234233011222006935064163542965827300151276013 32...
- 0.24257431785781985397922441858376486736156311385549281450394241750 60...
- 0.24474485438214052759717305161330543366386588708085807333992 8...
- 0.24600369400712937477975290028099807844398...
- 0.24681709192591636885663146 80...
- 0.2473814067574507777...
- 0.247793229299...
- 0.2482...

Singularity Structure – Av(4123,4231,4312)

Dominant singularity at

$$x = 0.22379330134707544472559413080653078228254418356103196904827043215739850630915030$$

with critical exponent

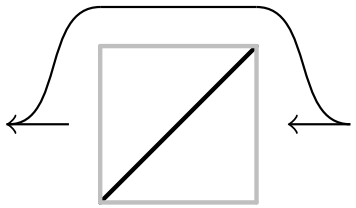
$$-1.00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000$$

Other singularities:

- 0.23806375980802816276234233011222006935064163542965827300151276013 32...
- 0.24257431785781985397922441858376486736156311385549281450394241750 60...
- 0.24474485438214052759717305161330543366386588708085807333992 8...
- 0.24600369400712937477975290028099807844398...
- 0.24681709192591636885663146 80...
- 0.2473814067574507777...
- 0.247793229299...
- 0.2482...

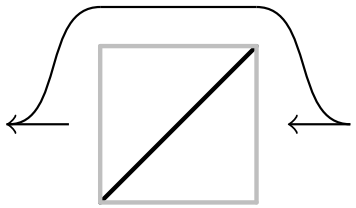
$$\Longrightarrow \sim C(4.468408...)^n$$

State Compression

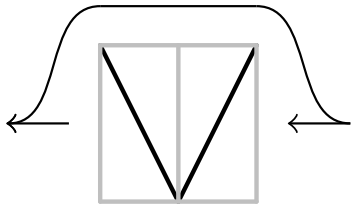


4^n permutations $\longrightarrow$ n (1 integer)

State Compression

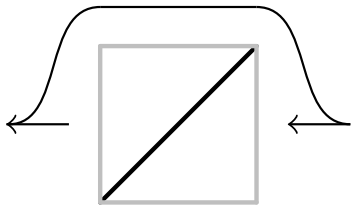


4^n permutations $\longrightarrow$ n (1 integer)

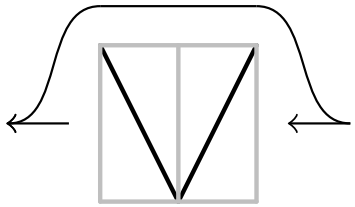


$5.828^n \longrightarrow 2^n \longrightarrow n$ (1 integer)

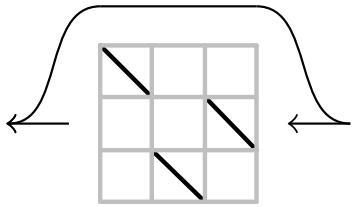
State Compression



4^n permutations $\longrightarrow$ n (1 integer)

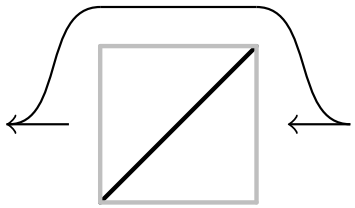


$5.828^n \longrightarrow 2^n \longrightarrow n$ (1 integer)

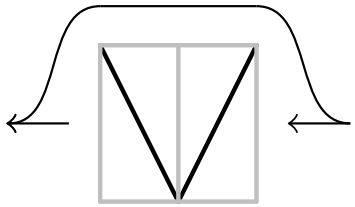


$4.977^n \longrightarrow n^3$ (3 integers)

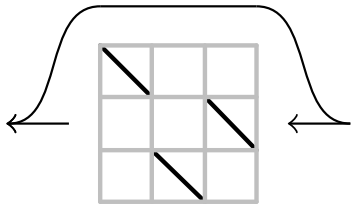
State Compression



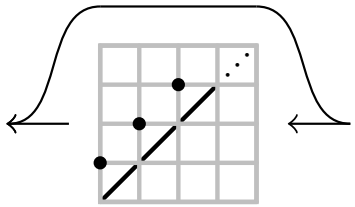
4^n permutations $\longrightarrow$ n (1 integer)



$5.828^n \longrightarrow 2^n \longrightarrow n$ (1 integer)



$4.977^n \longrightarrow n^3$ (3 integers)



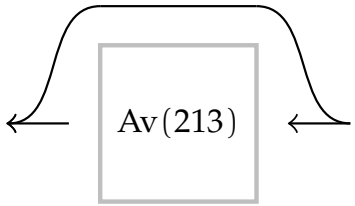
$5.892^n \longrightarrow 2^n \longrightarrow n^3$ (3 integers)

State Compression

$$11.6^n(?) \longrightarrow 4^n \longrightarrow ???$$

State Compression

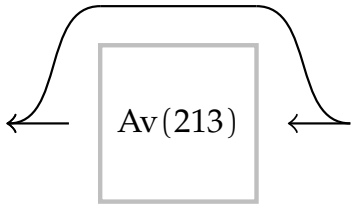
$$11.6^n(?) \longrightarrow 4^n \longrightarrow ???$$



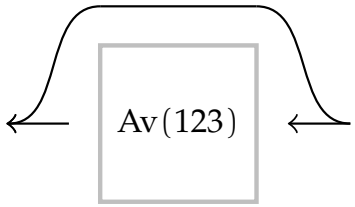
$$8^n \longrightarrow 4^n \longrightarrow ???$$

State Compression

$11.6^n(?) \longrightarrow 4^n \longrightarrow ???$



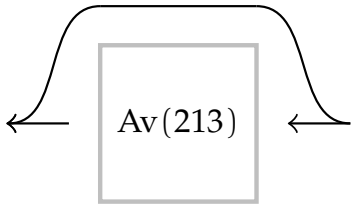
$8^n \longrightarrow 4^n \longrightarrow ???$



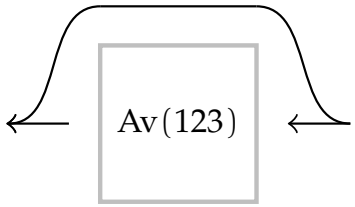
$9^n \longrightarrow 4^n \longrightarrow ???$

State Compression

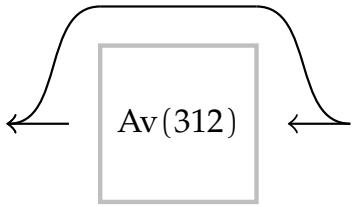
$11.6^n(?) \longrightarrow 4^n \longrightarrow ???$



$8^n \longrightarrow 4^n \longrightarrow ???$

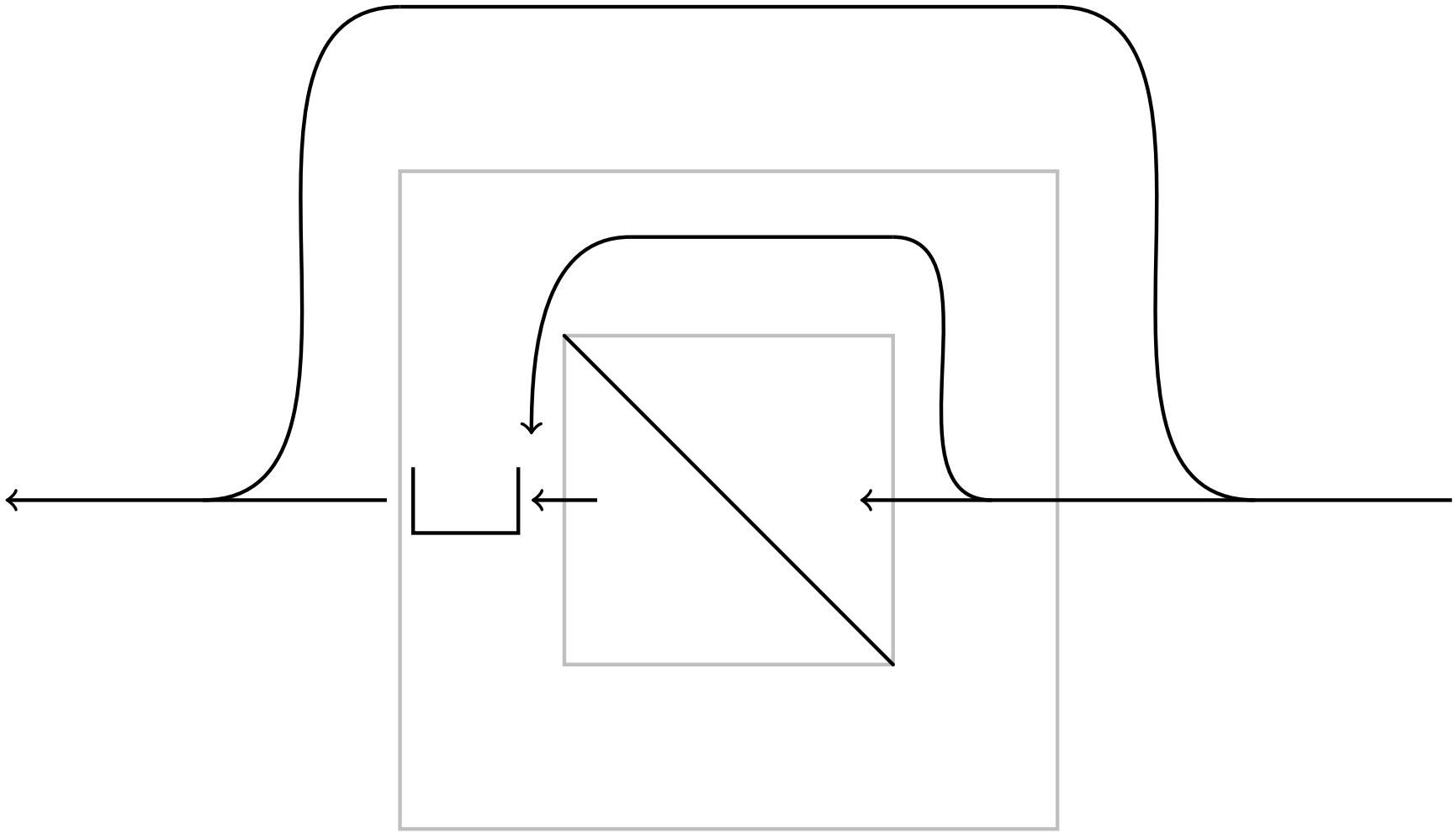


$9^n \longrightarrow 4^n \longrightarrow ???$



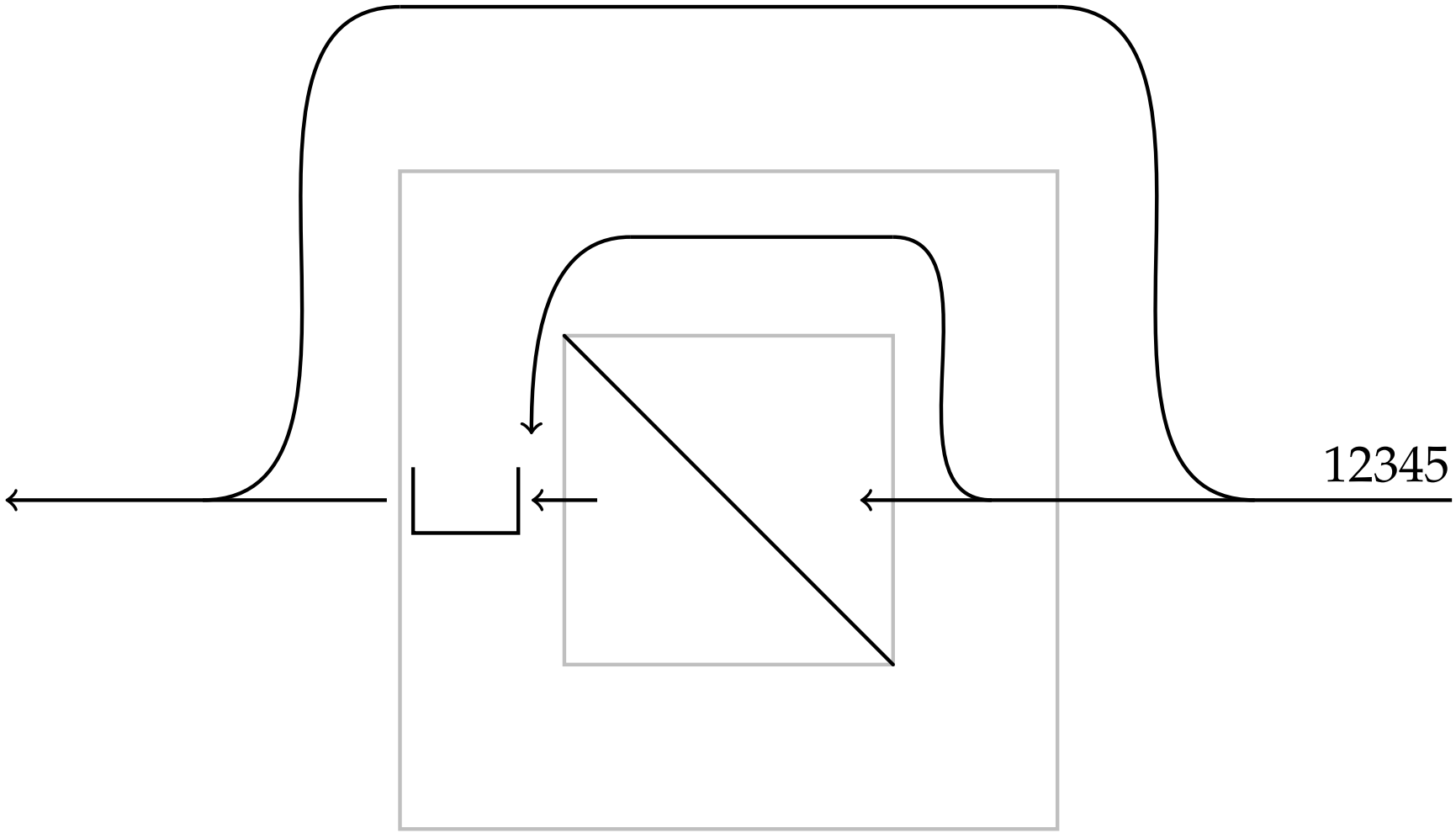
$9^n \longrightarrow 4^n \longrightarrow n^2$ (2 integers) !!!

Nested Av(12)-Machine



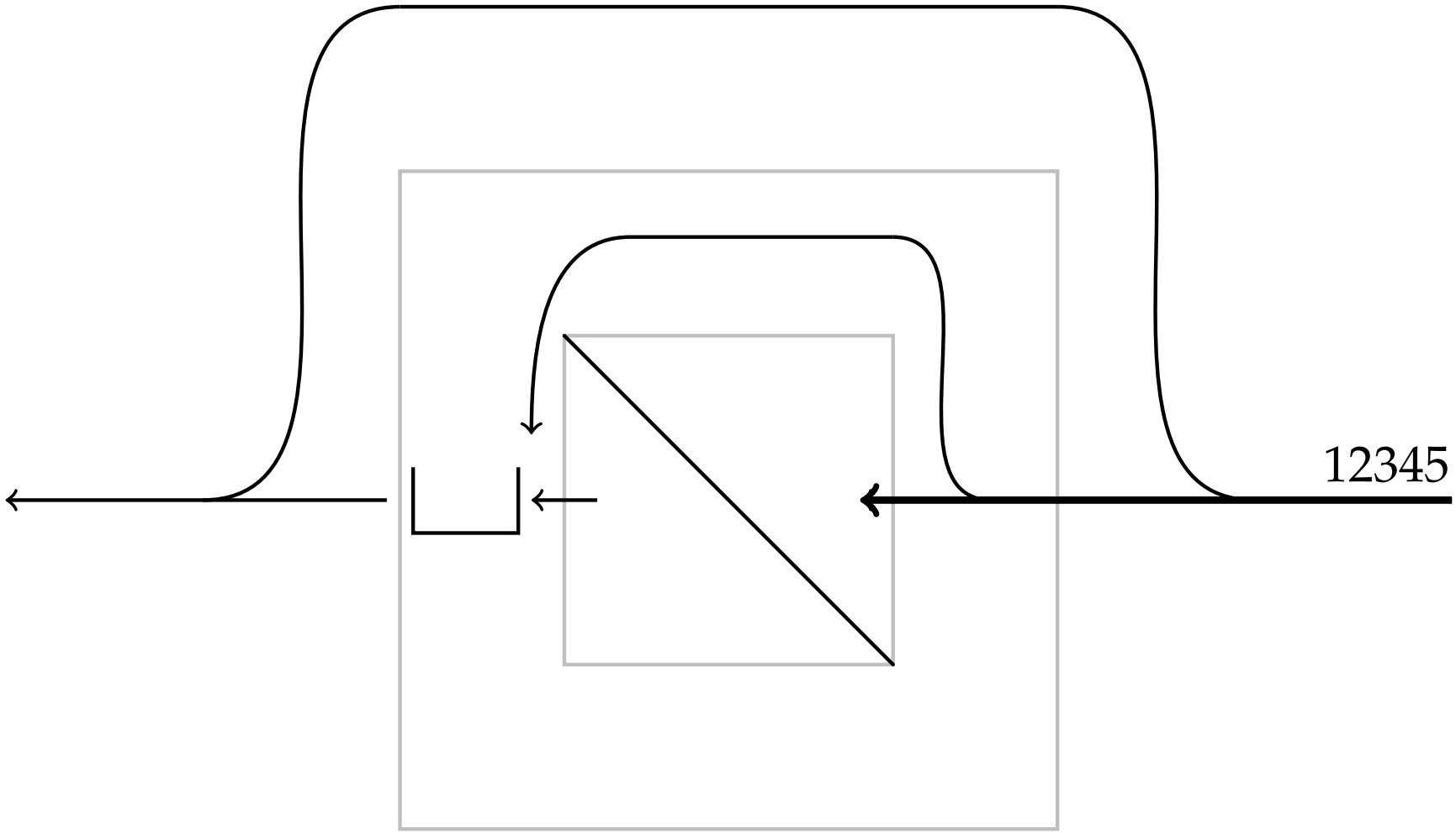
Generates Av(4312)

Nested Av(12)-Machine



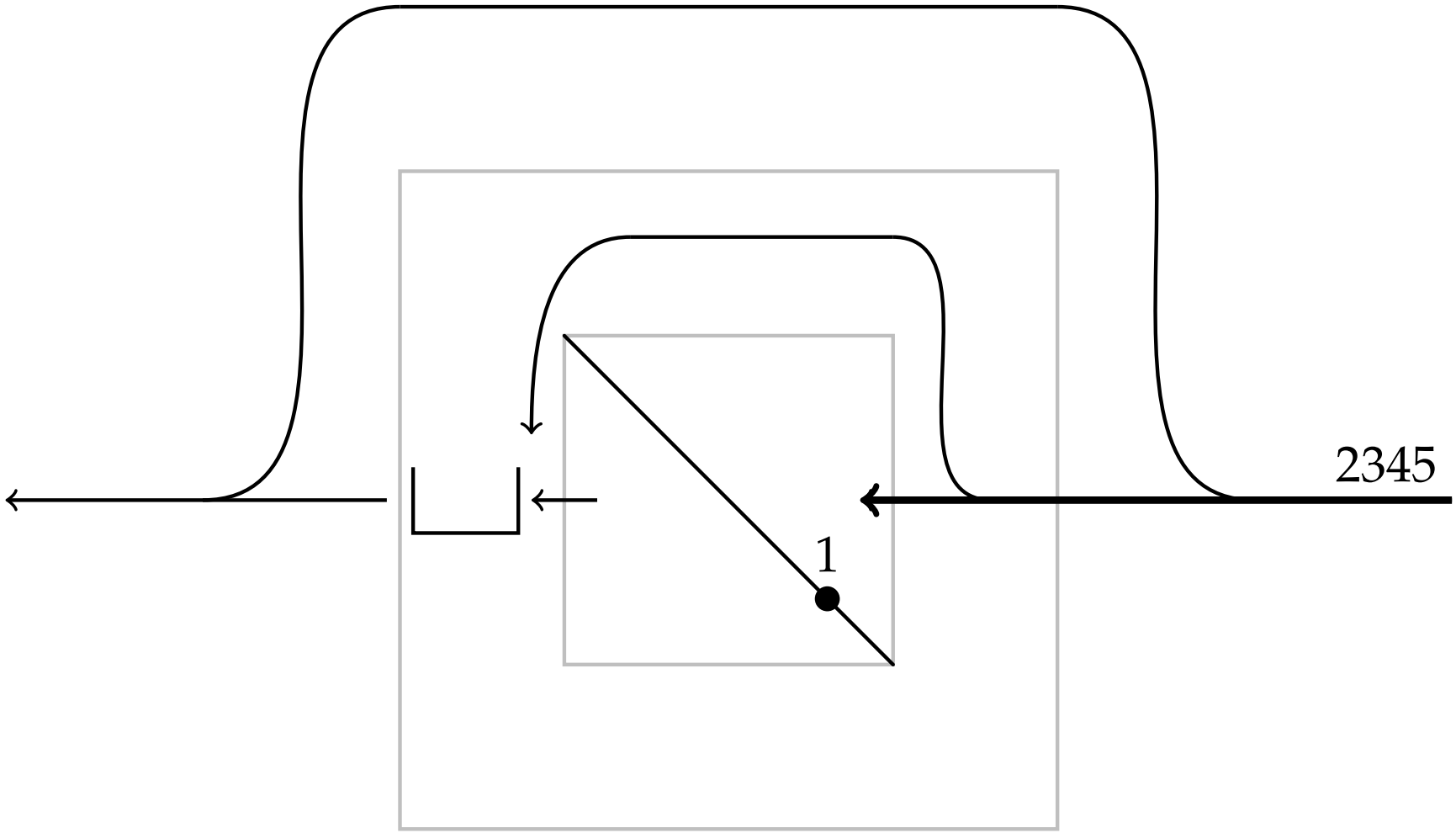
Generates Av(4312)

Nested Av(12)-Machine



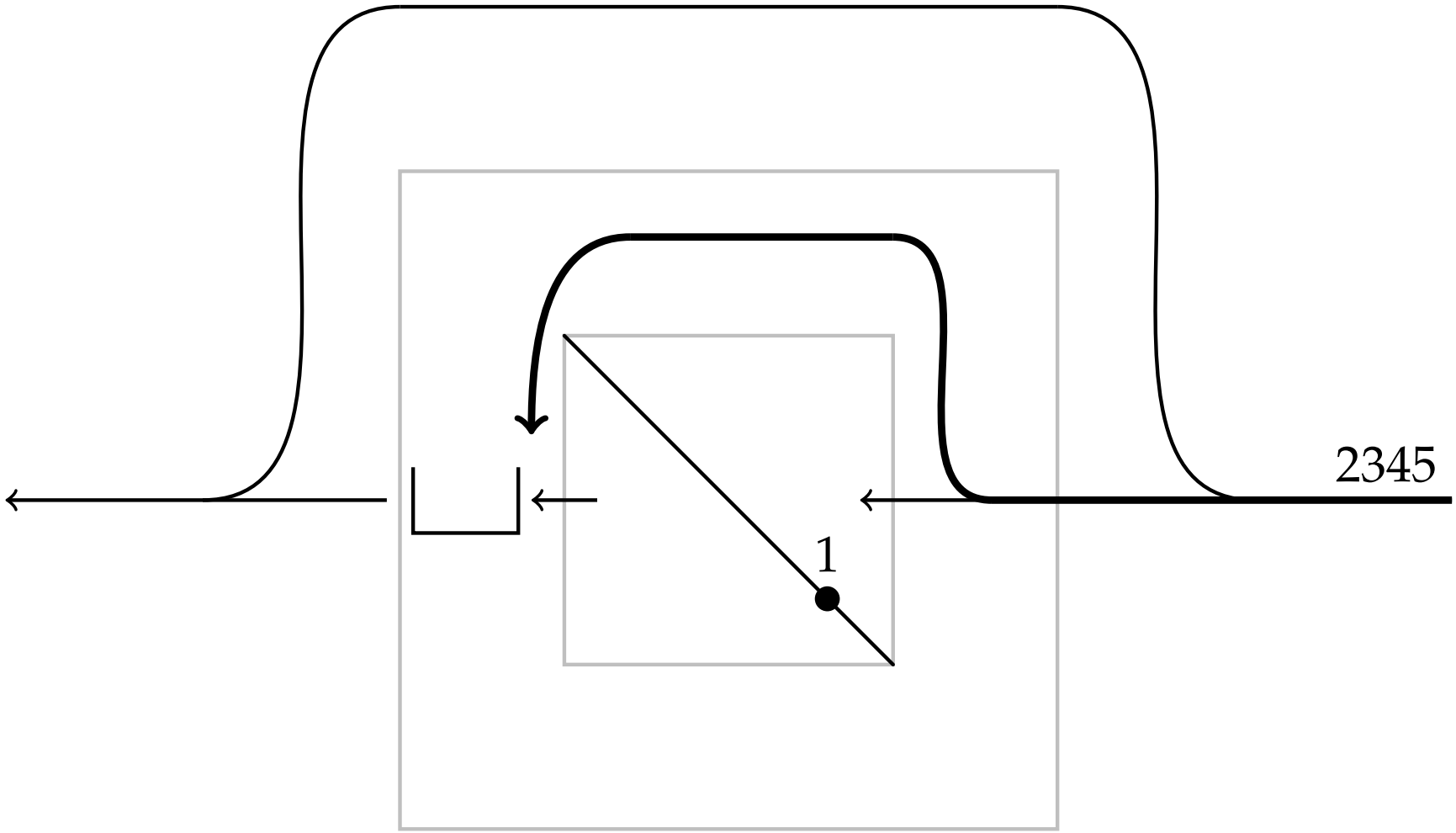
Generates Av(4312)

Nested Av(12)-Machine



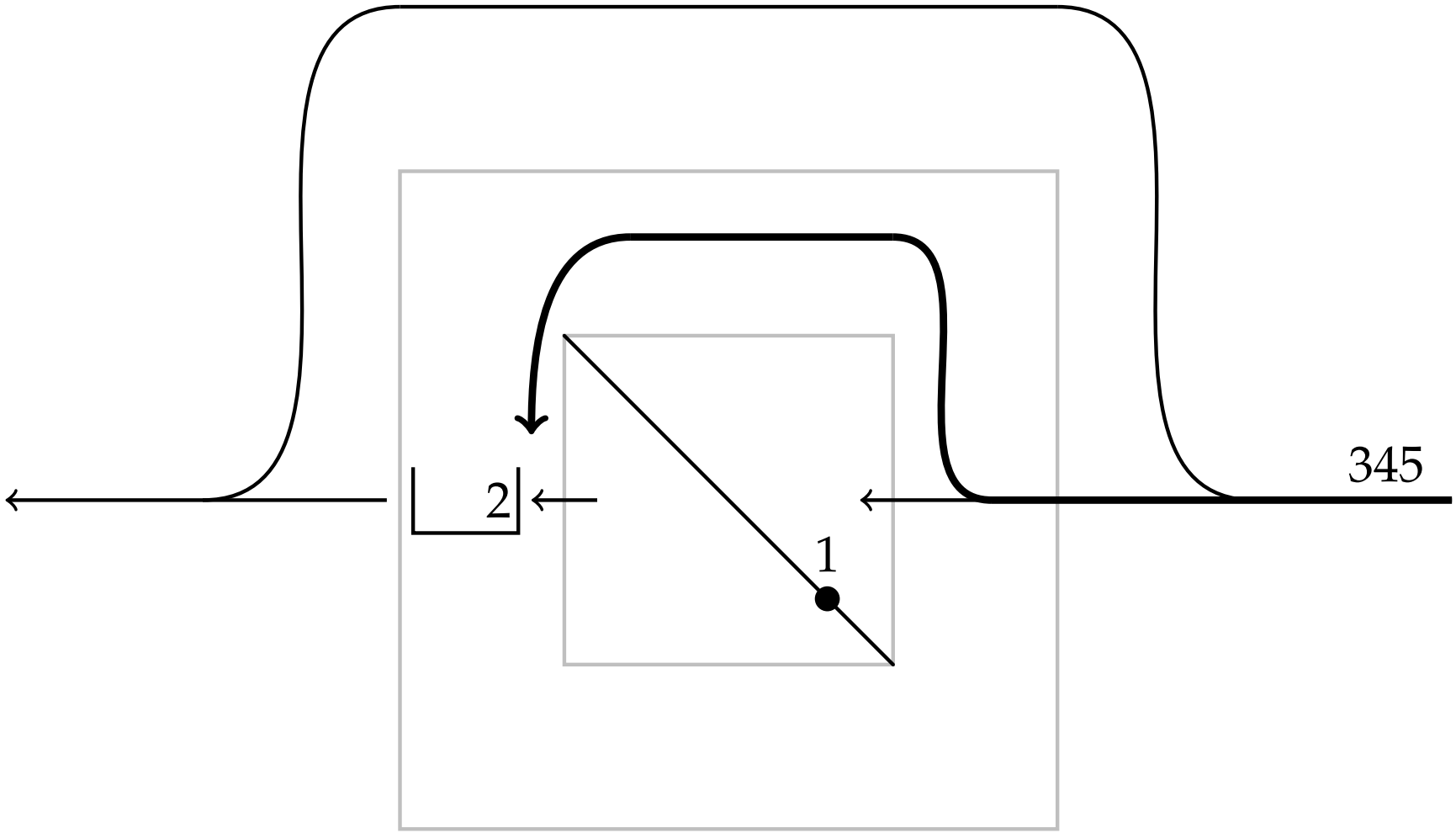
Generates Av(4312)

Nested Av(12)-Machine



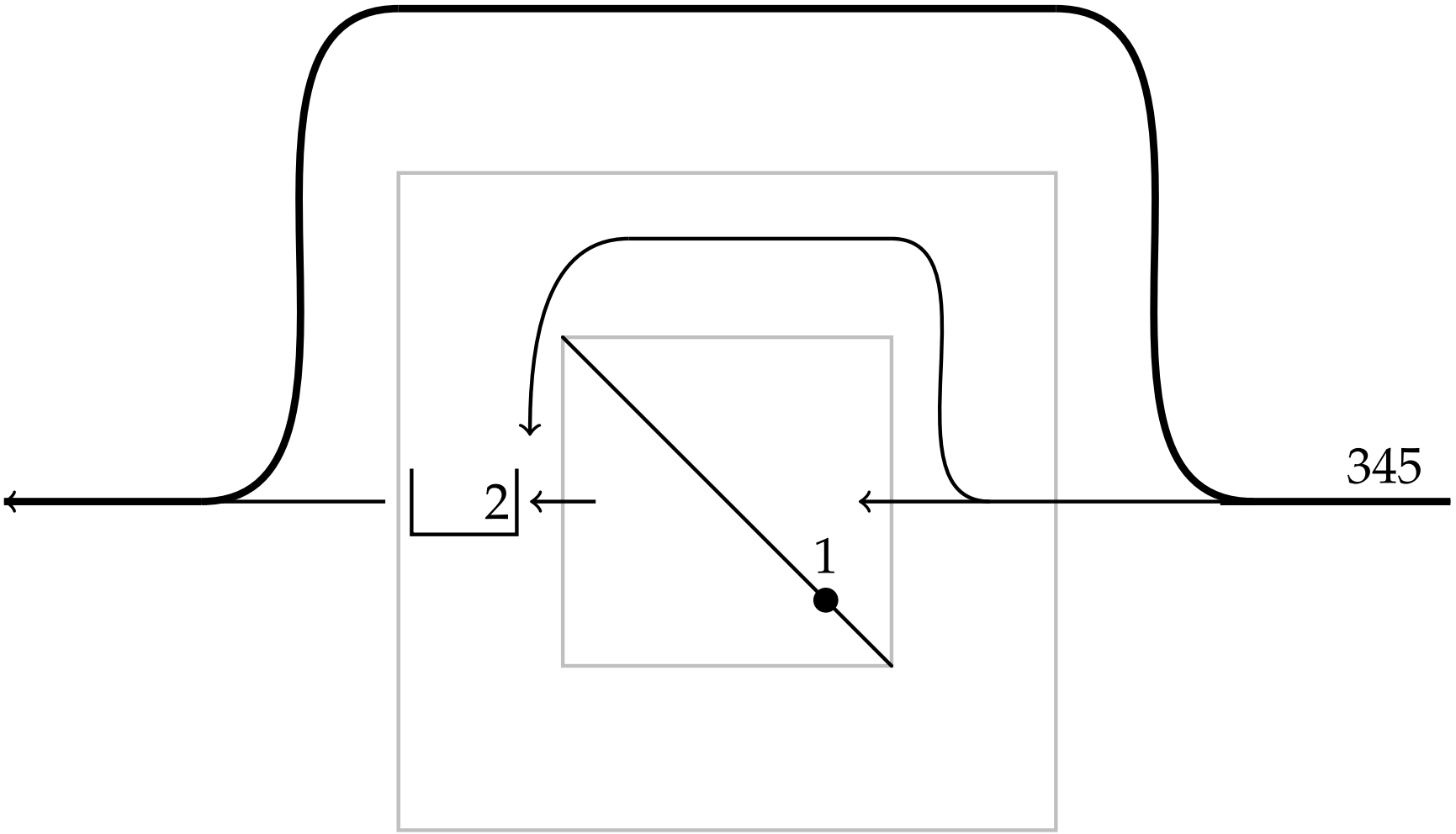
Generates Av(4312)

Nested Av(12)-Machine



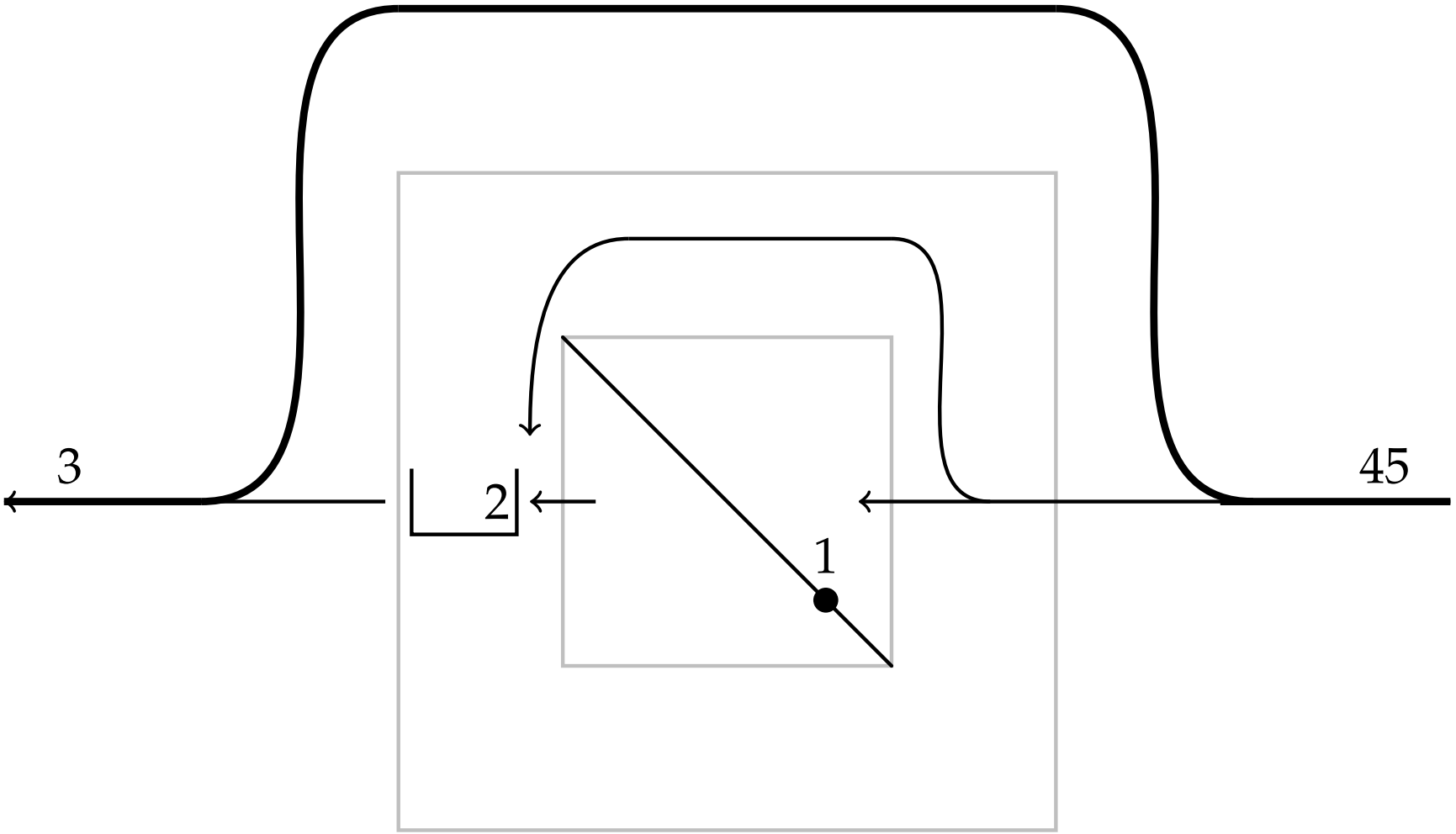
Generates Av(4312)

Nested Av(12)-Machine



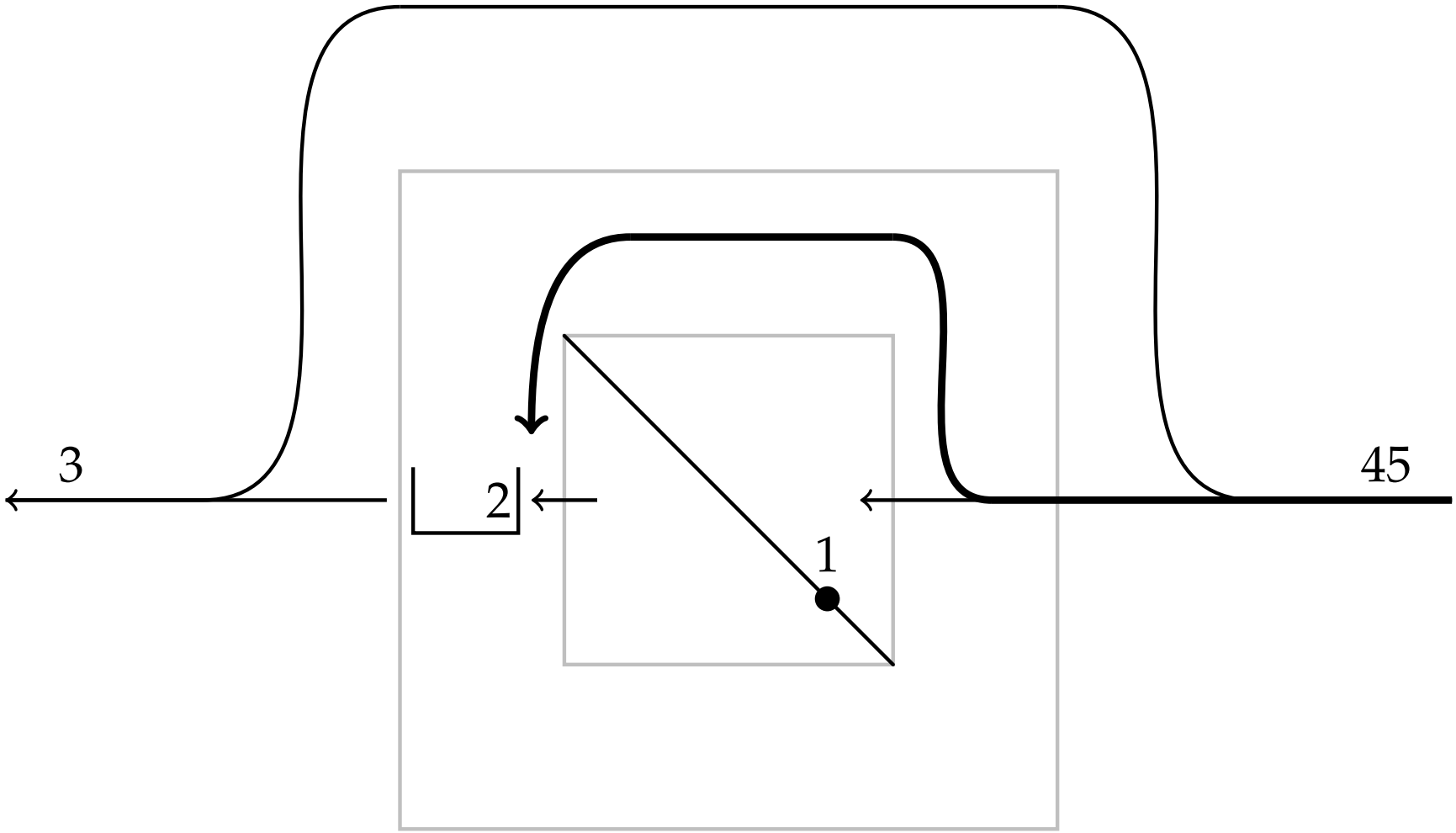
Generates Av(4312)

Nested Av(12)-Machine



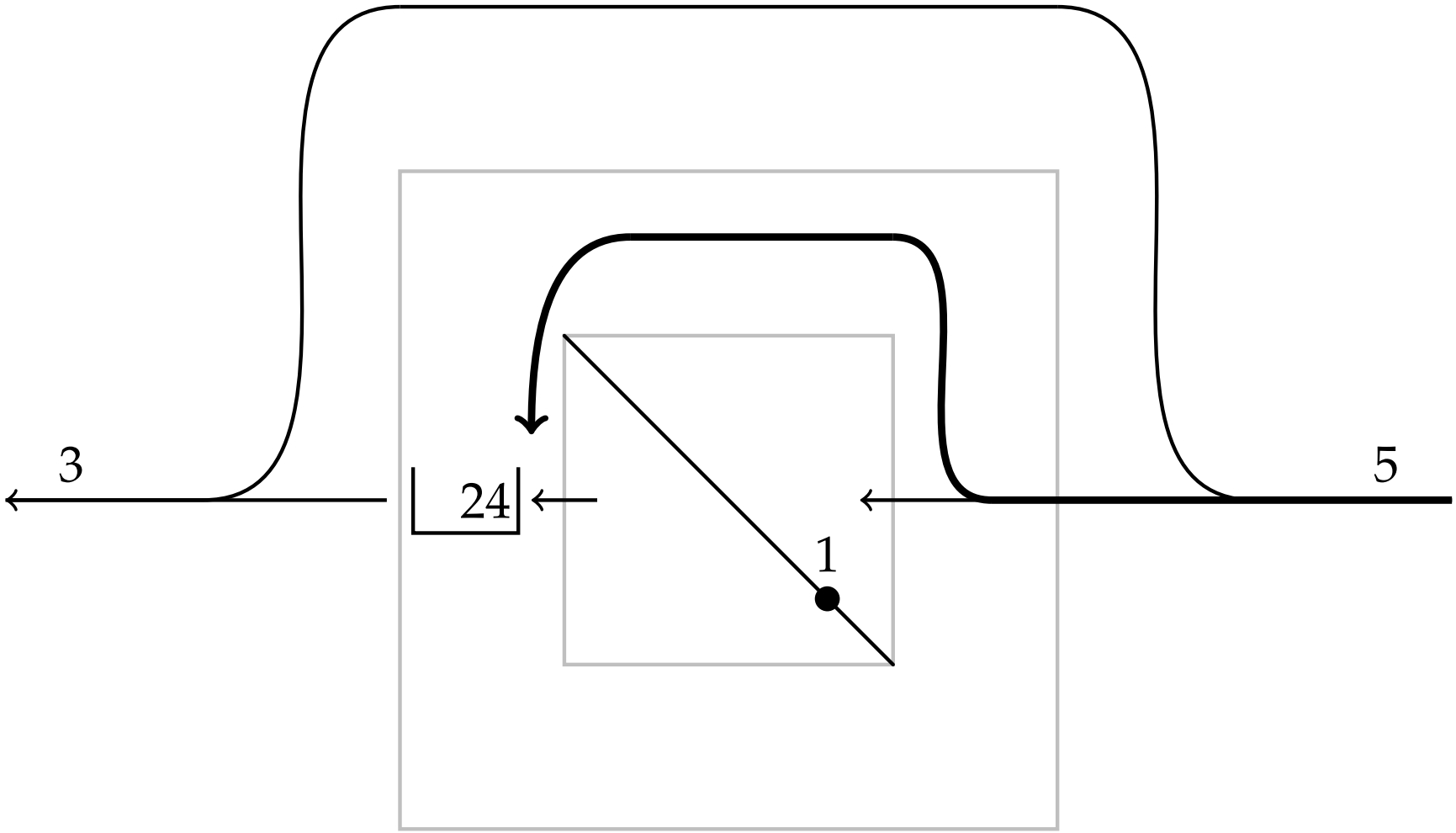
Generates Av(4312)

Nested Av(12)-Machine



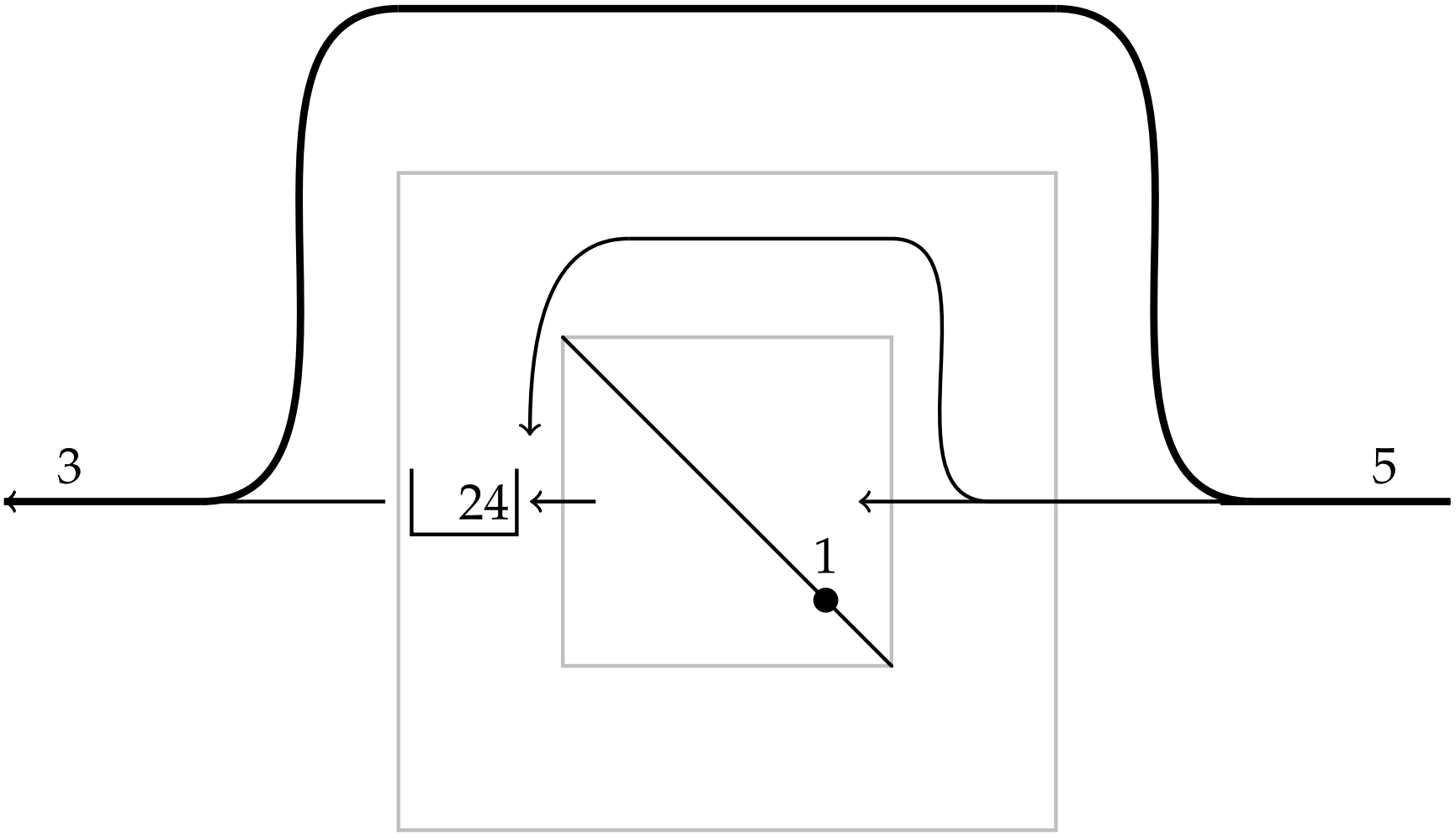
Generates Av(4312)

Nested Av(12)-Machine



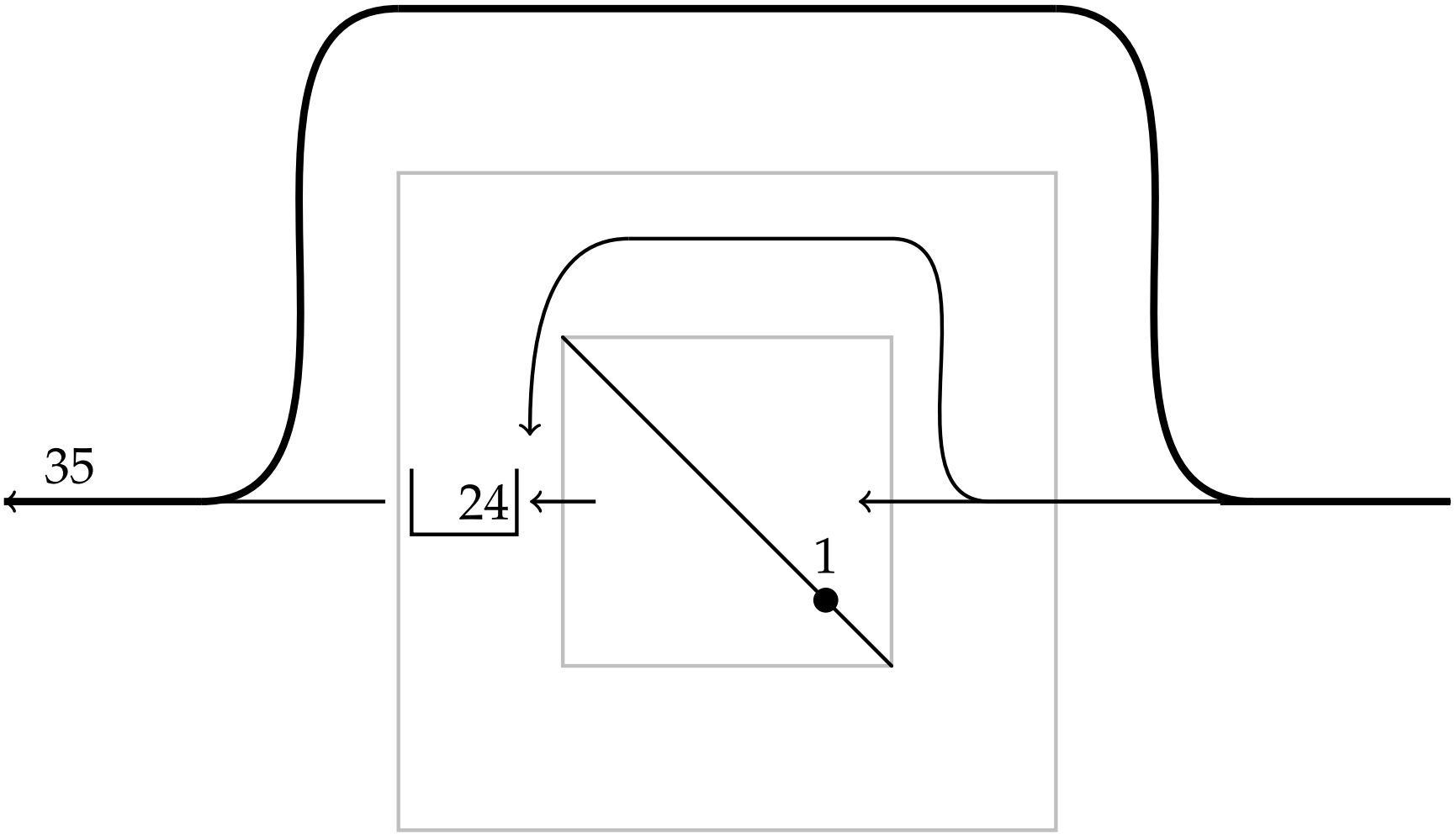
Generates Av(4312)

Nested Av(12)-Machine



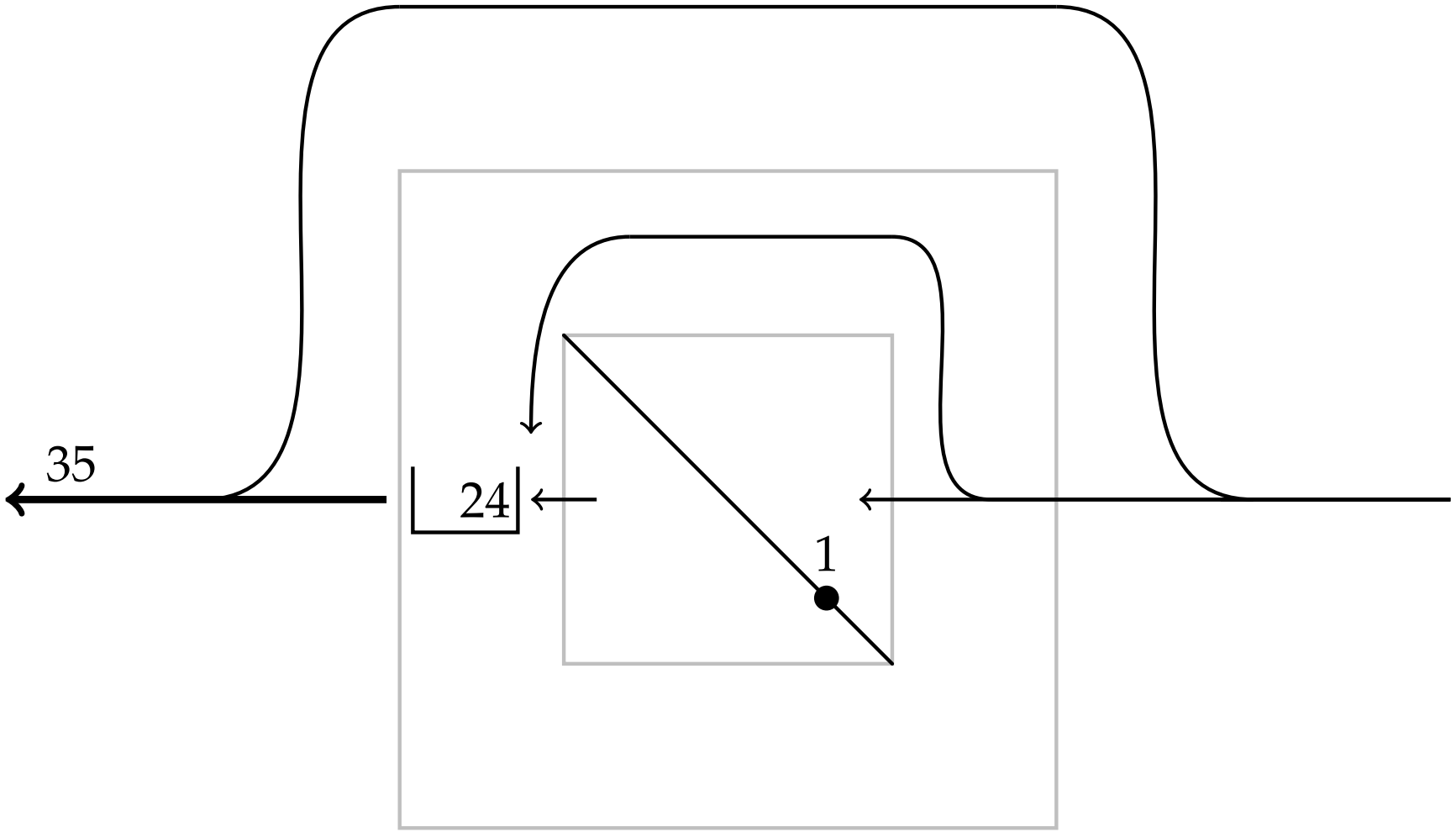
Generates Av(4312)

Nested Av(12)-Machine



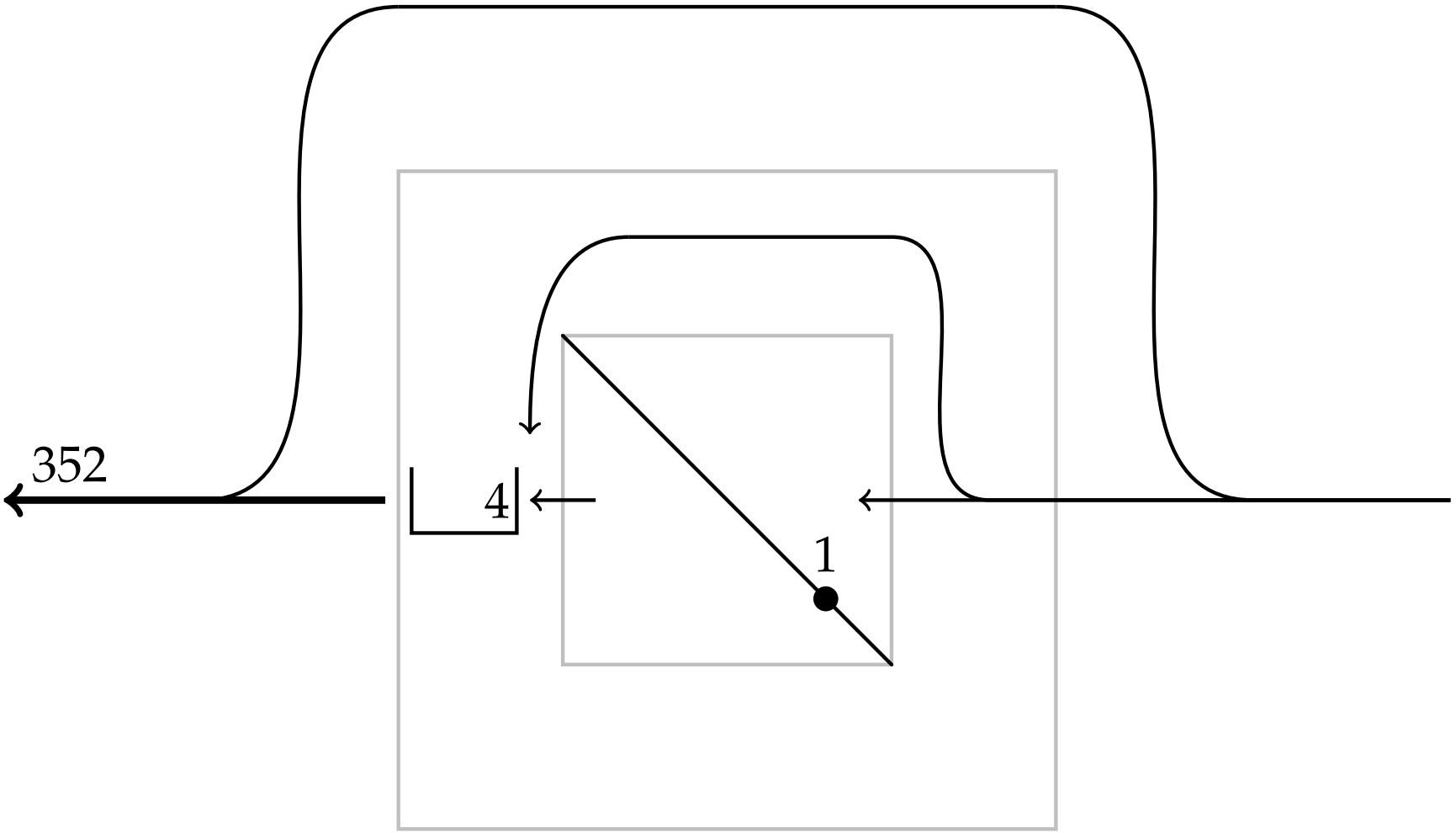
Generates Av(4312)

Nested Av(12)-Machine



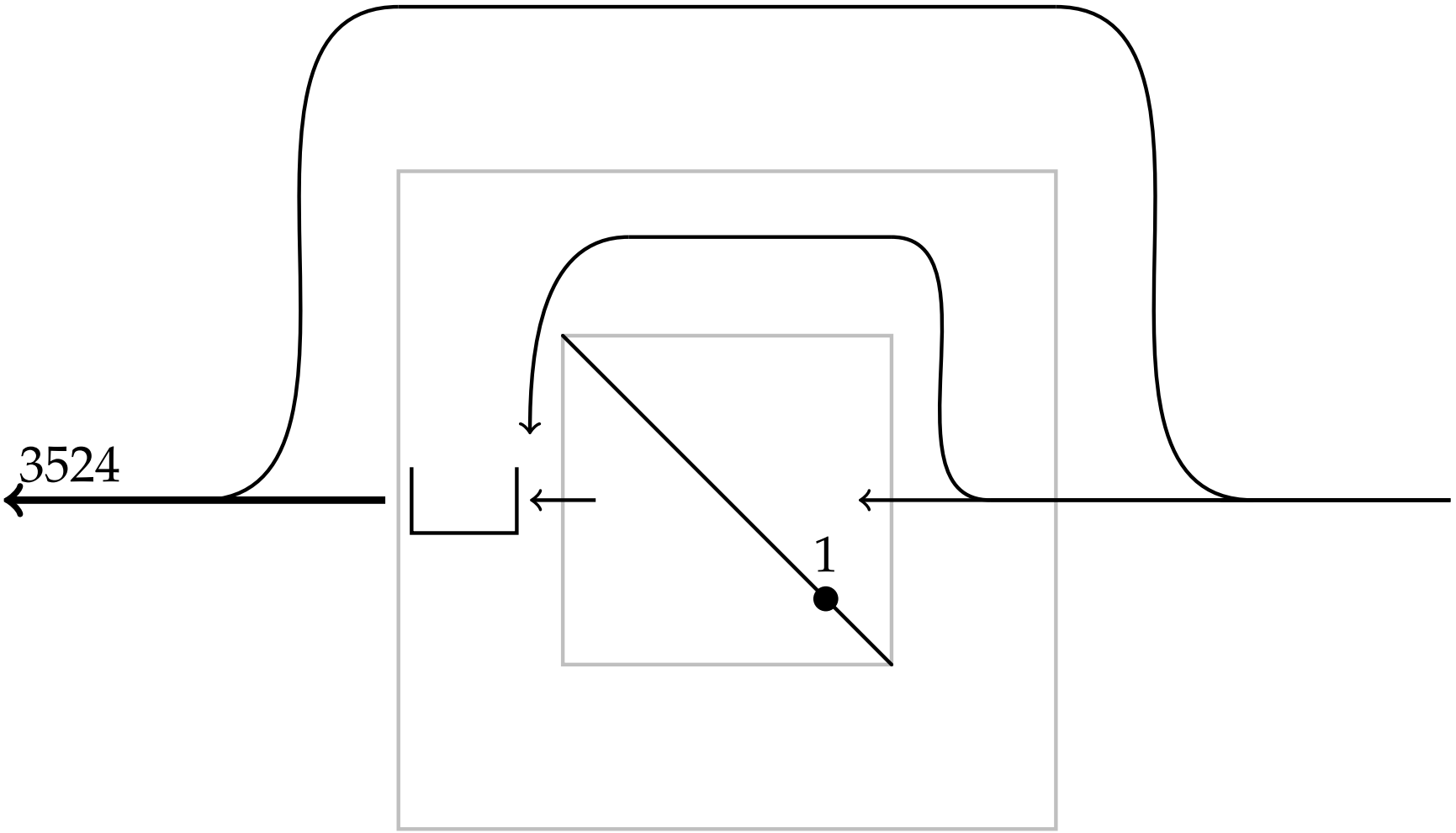
Generates Av(4312)

Nested Av(12)-Machine



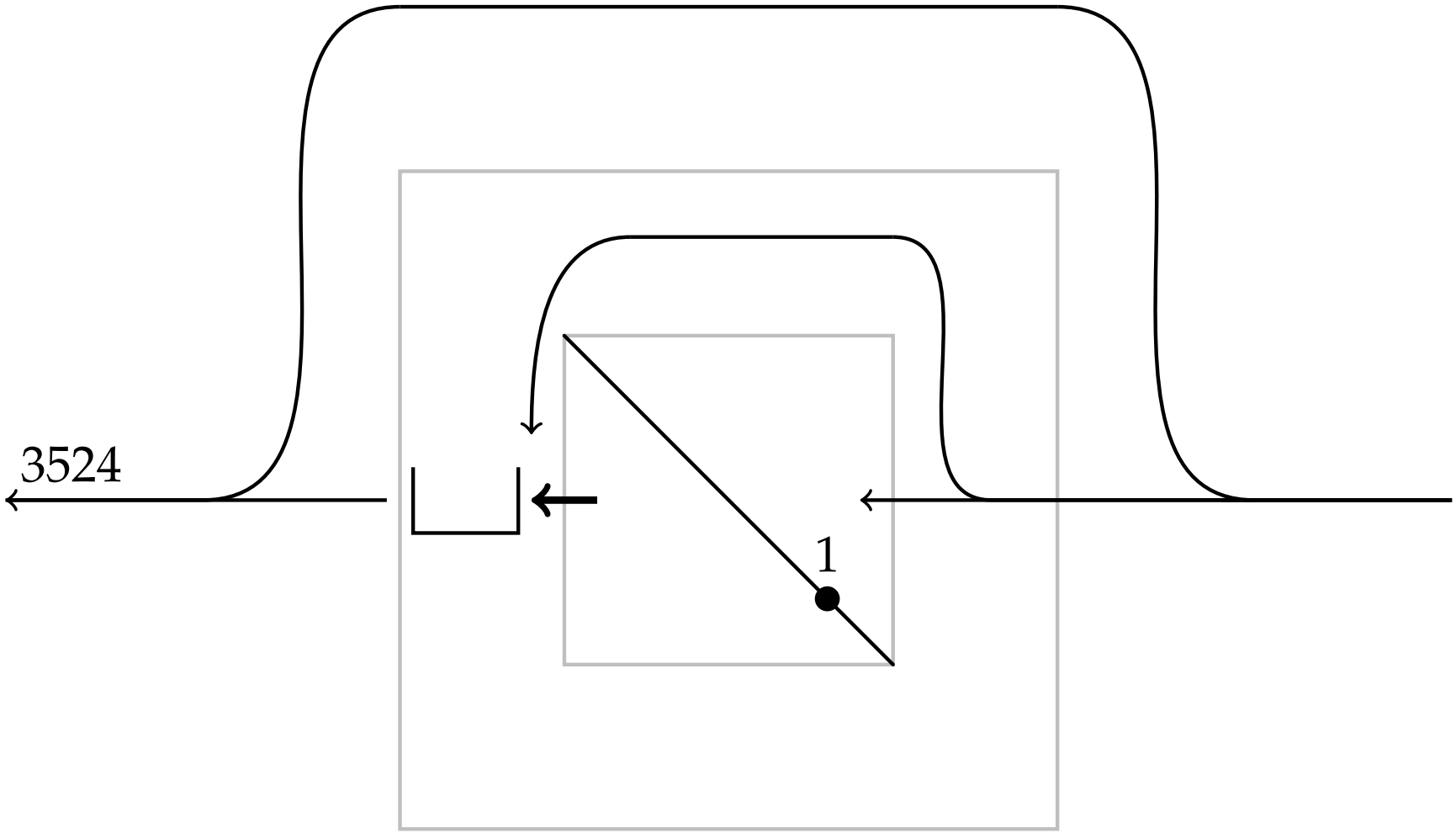
Generates Av(4312)

Nested Av(12)-Machine



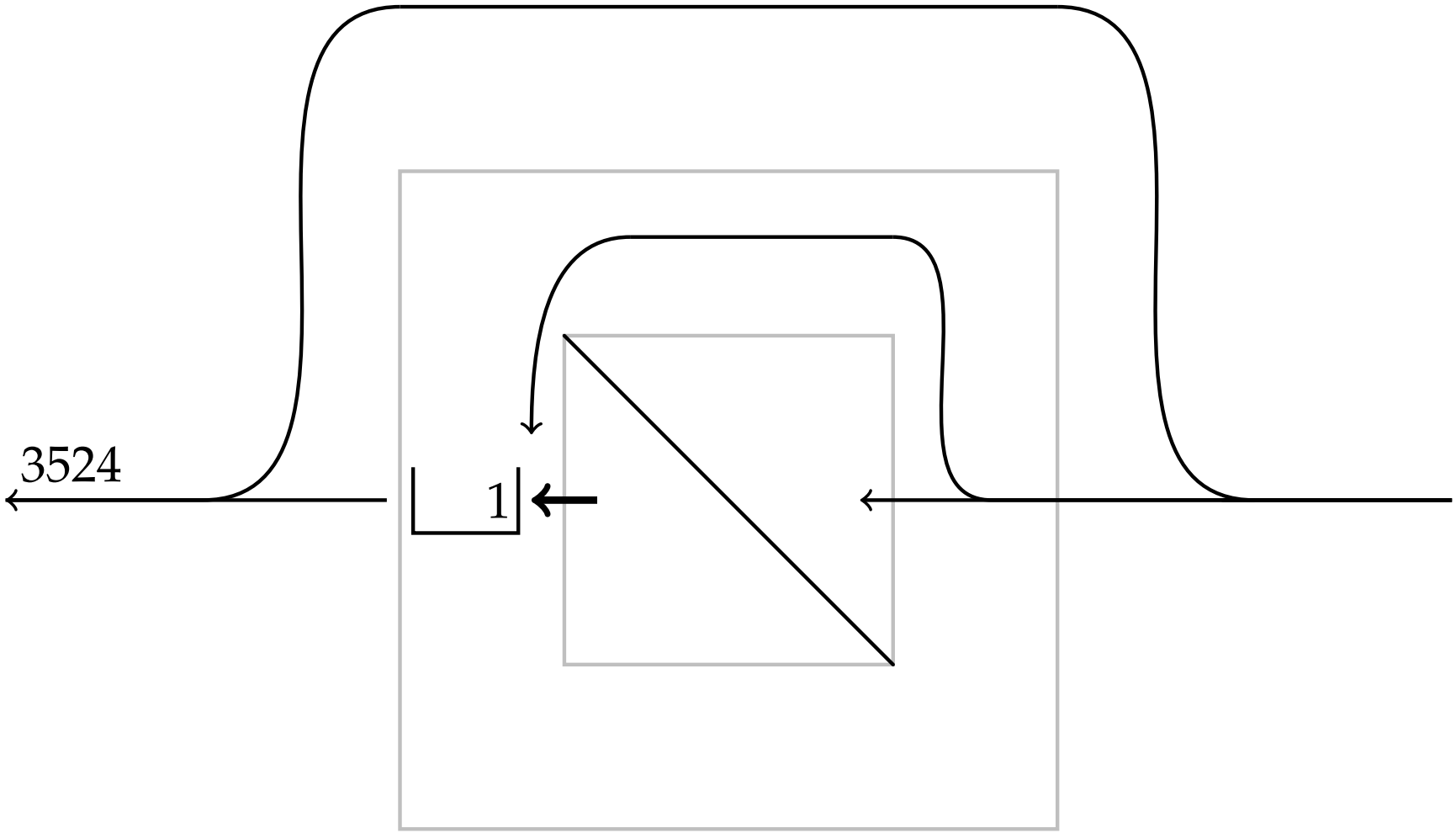
Generates Av(4312)

Nested Av(12)-Machine



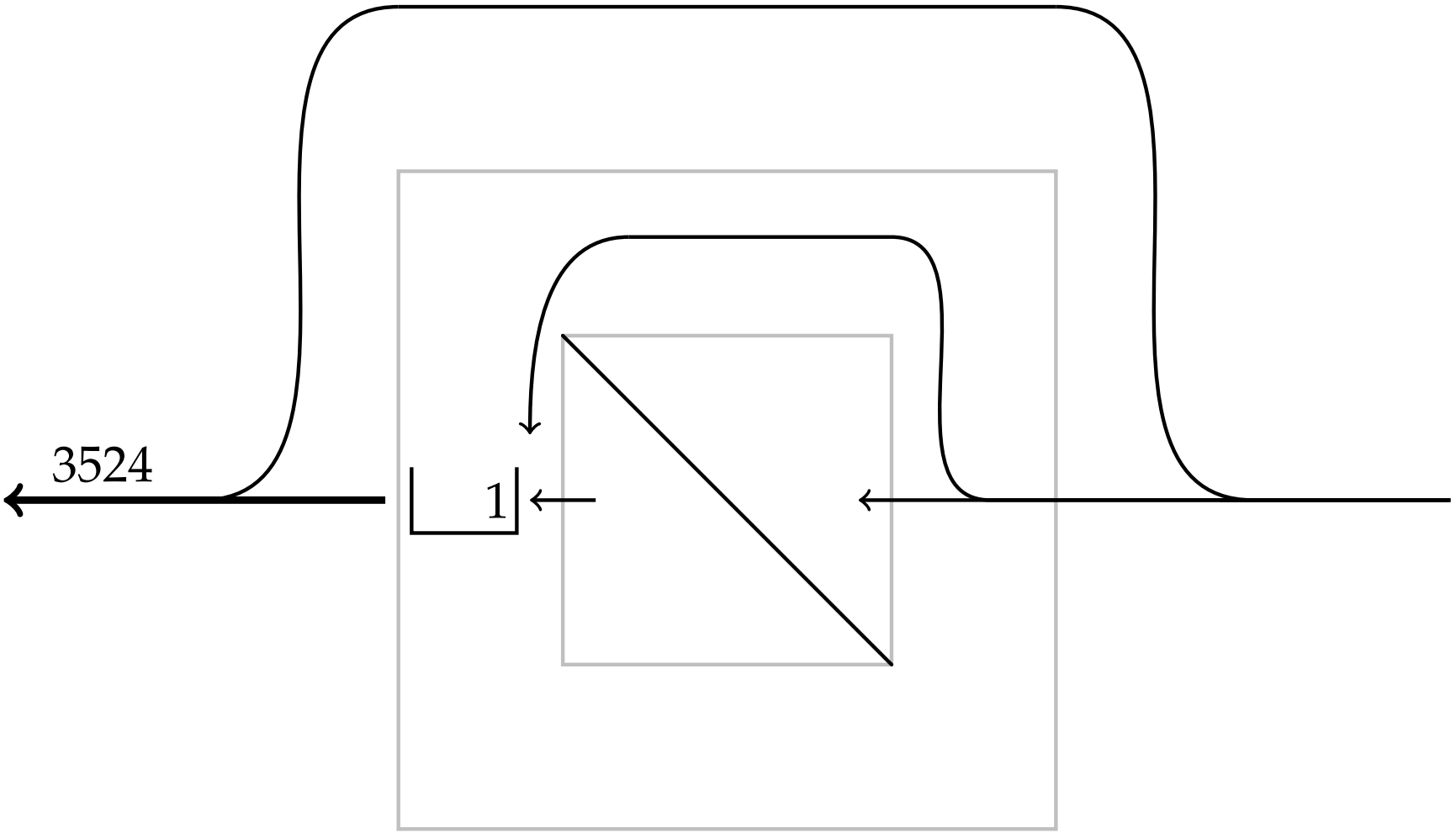
Generates Av(4312)

Nested Av(12)-Machine



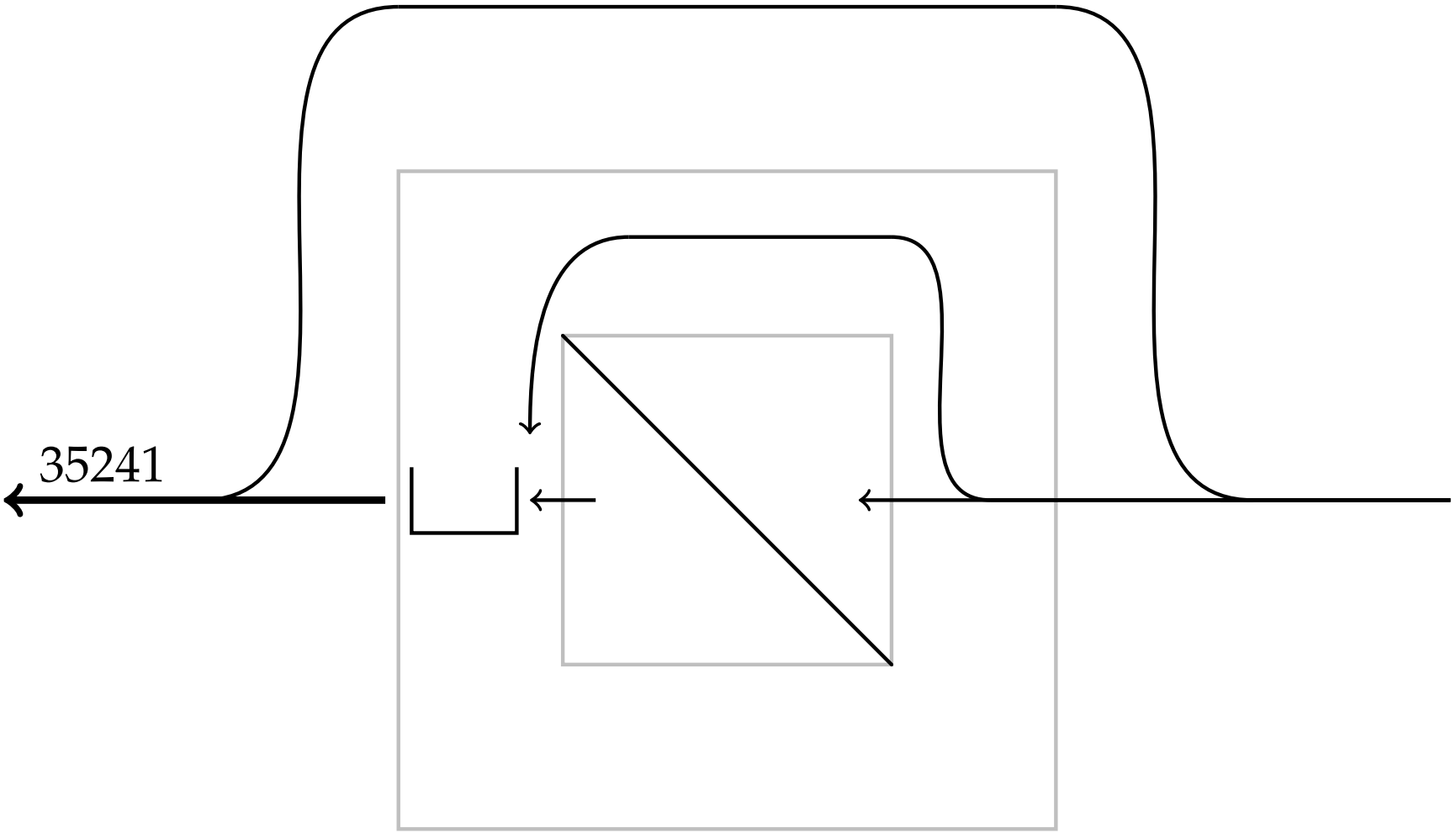
Generates Av(4312)

Nested Av(12)-Machine



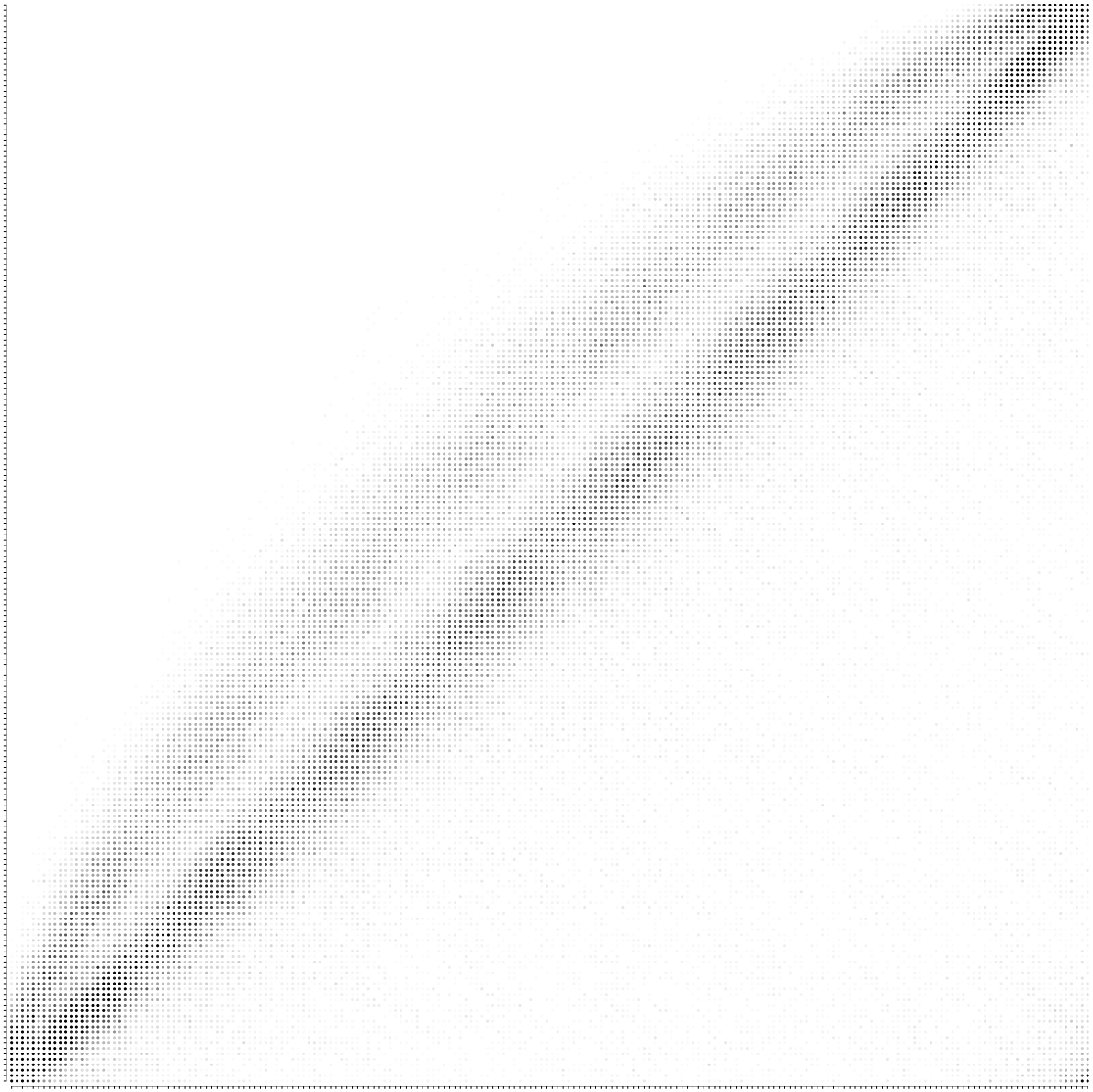
Generates Av(4312)

Nested Av(12)-Machine

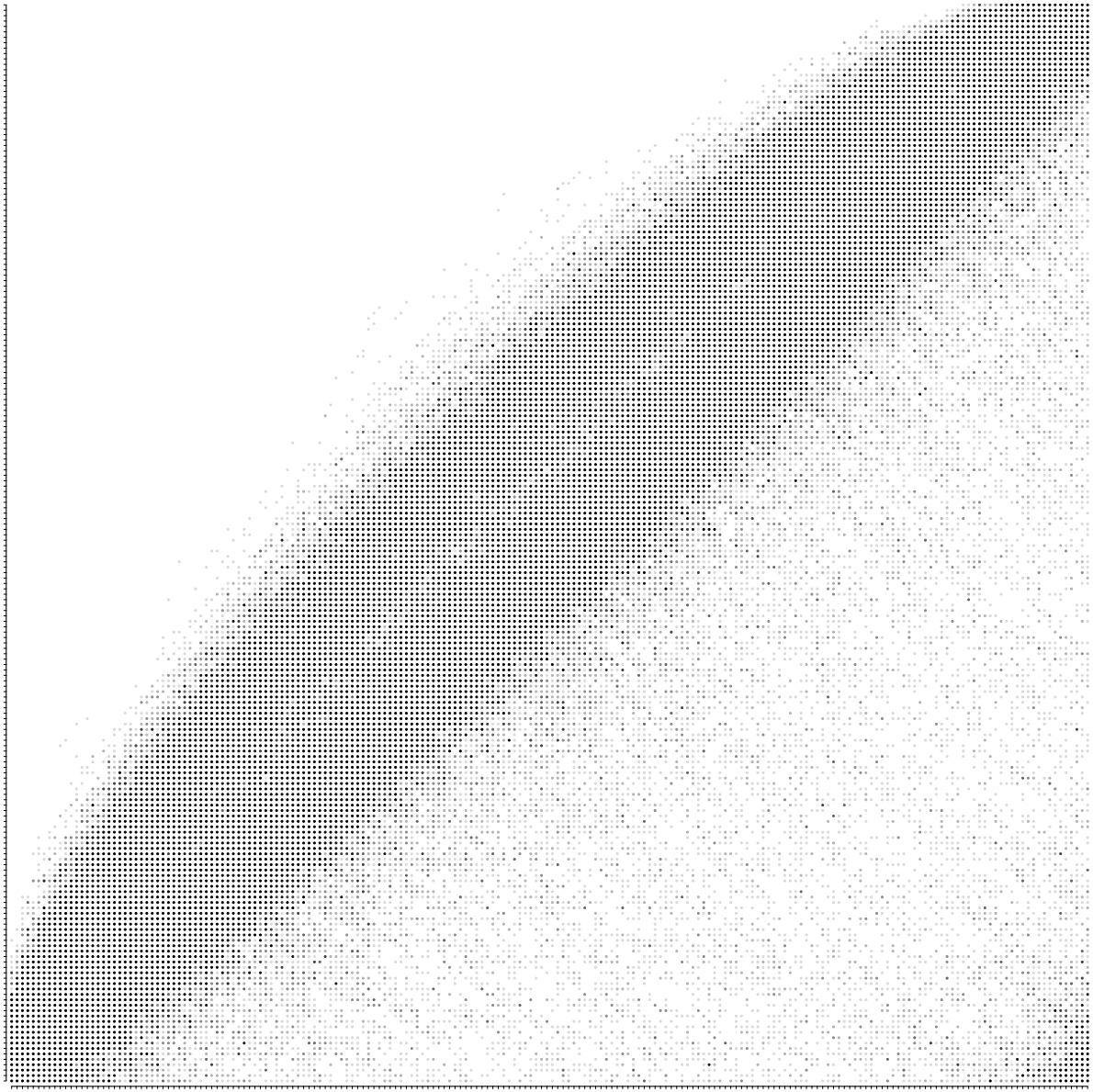


Generates Av(4312)

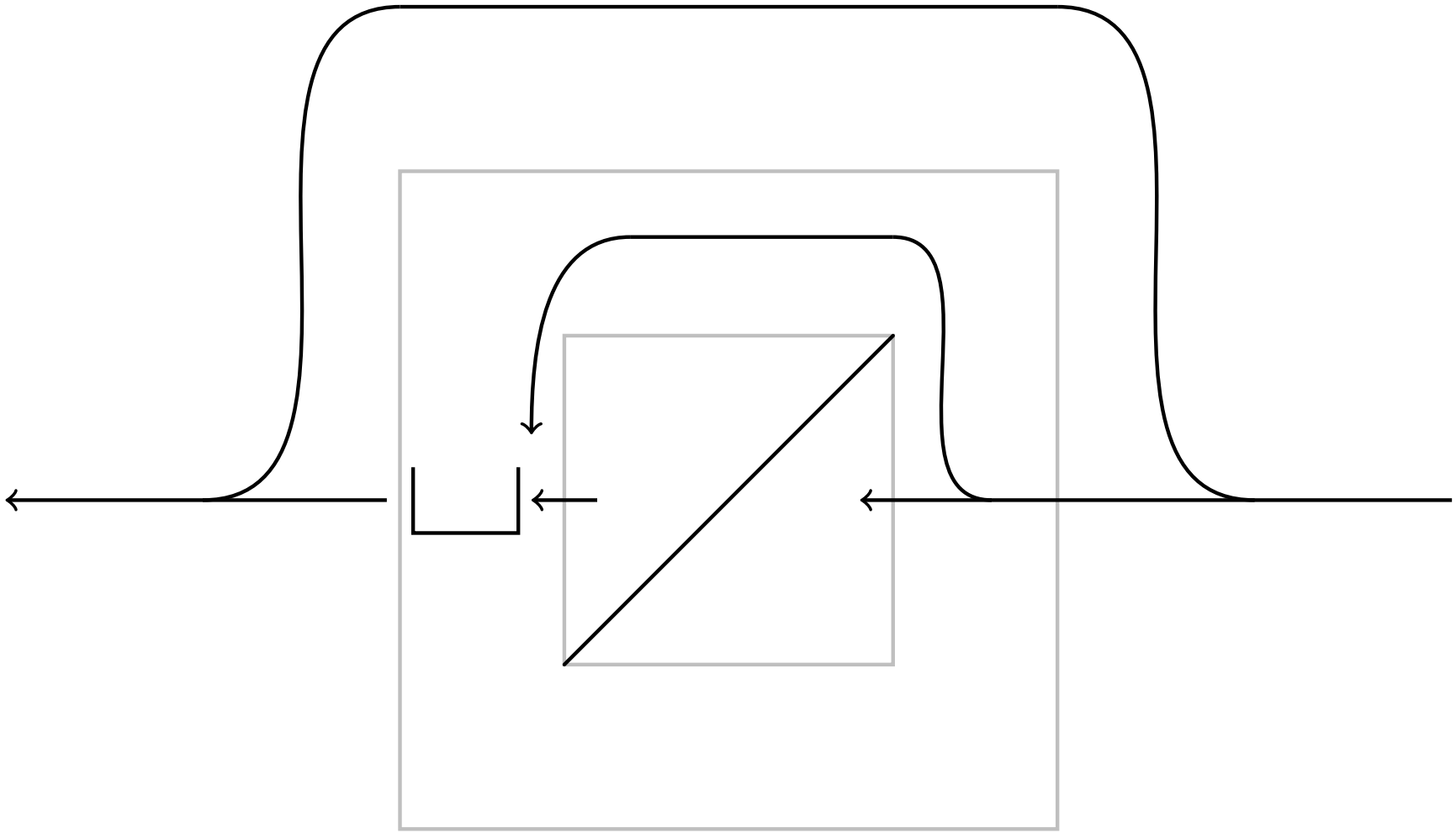
Random Sampling



Random Sampling

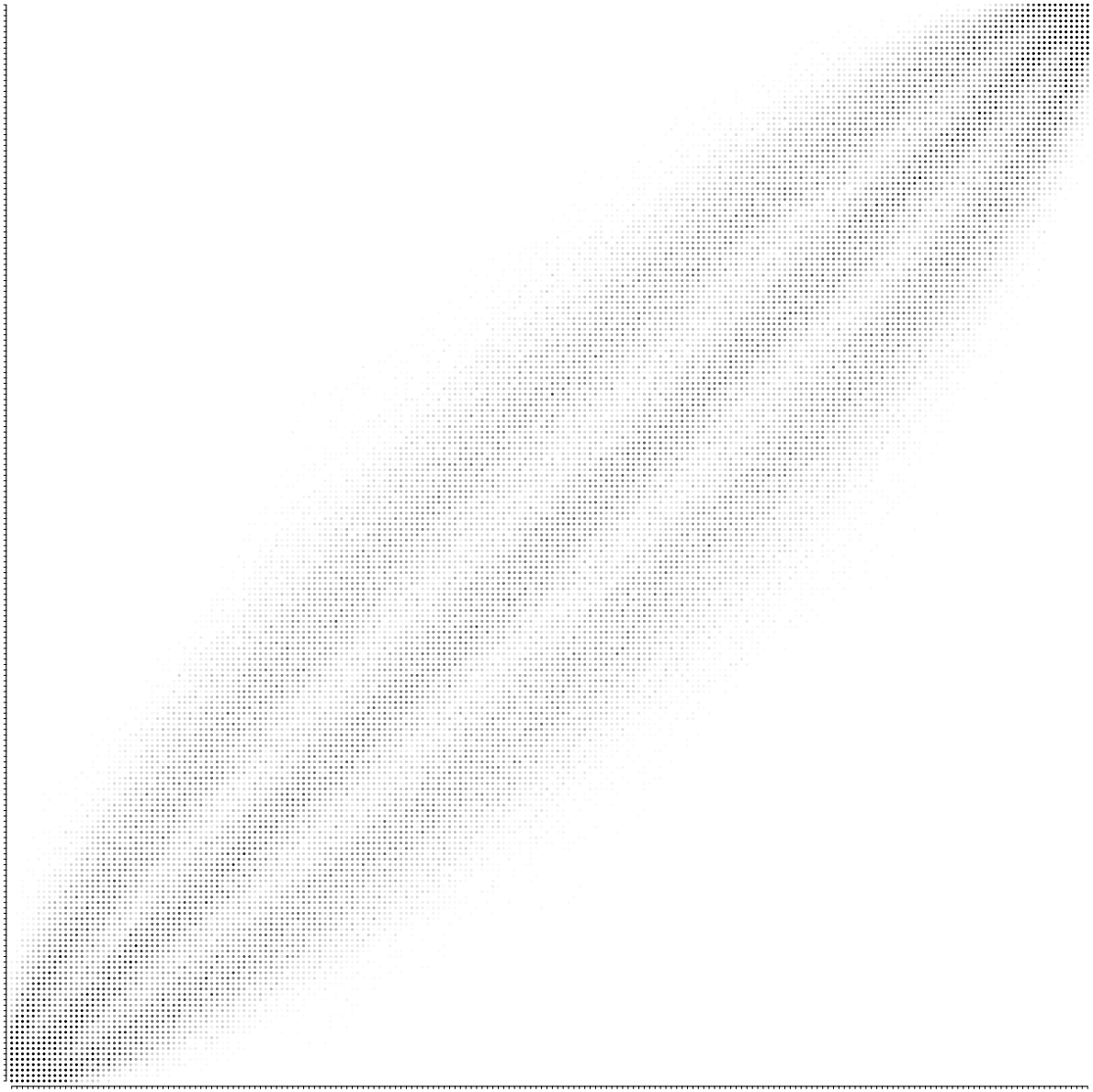


Nested Av(21)-Machine



Generates Av(4321) Free bijection to Av(4312)!

Random Sampling



What about Av(4231)?

Nesting doesn't work.

What about Av(4231)?

Nesting doesn't work.

Some heuristics suggest state compression isn't possible with a standard 𝒞-Machine.

What about Av(4231)?

Nesting doesn't work.

Some heuristics suggest state compression isn't possible with a standard 𝒞-Machine.

Perhaps a more complicated machine: second bypass, add a buffer or a stack, multiple moves at once, ...

Thanks!