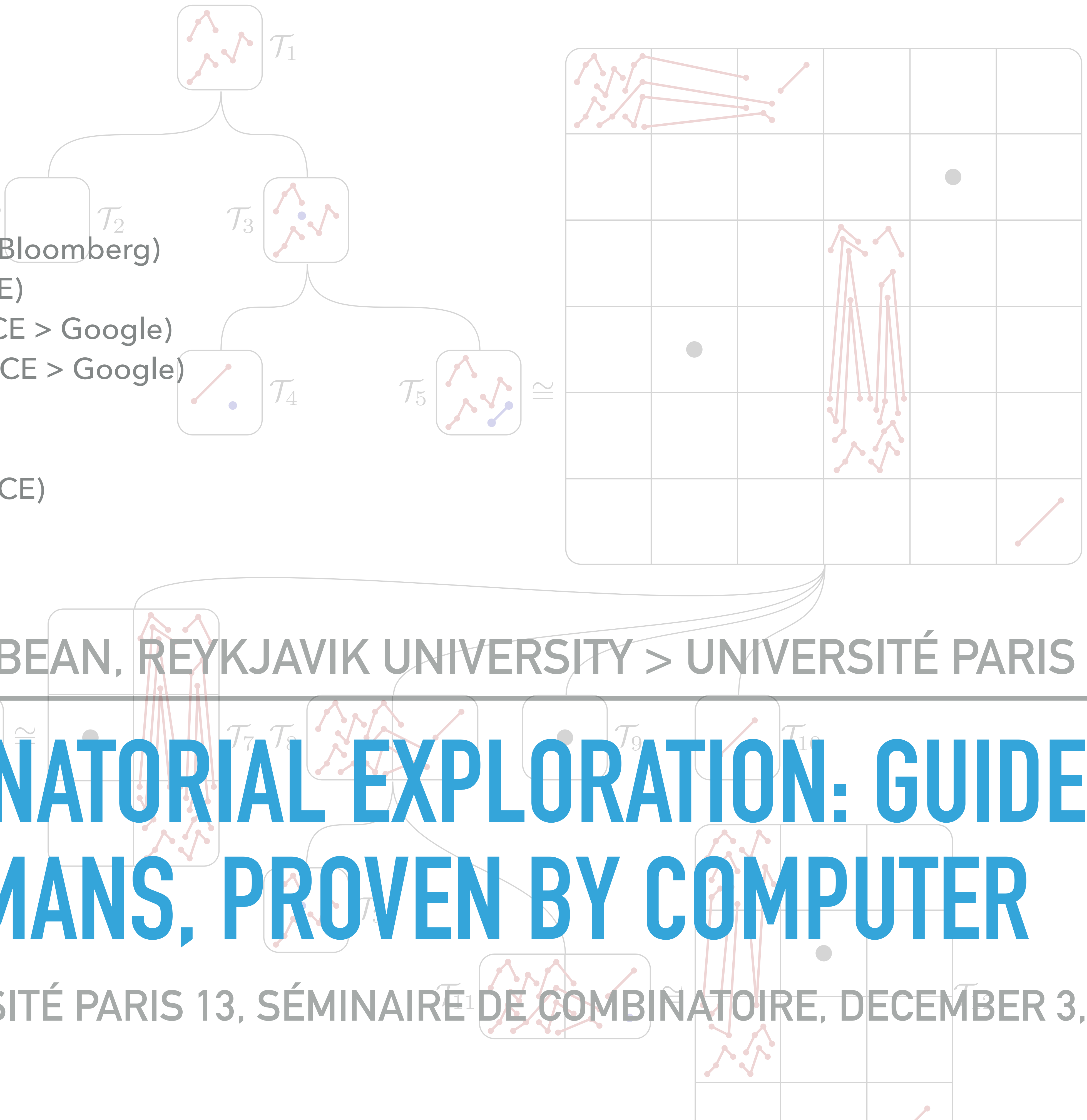


Joint work with
Michael Albert (NZ)
Arnar Arnarson (ICE)
Ragnar Árdal (ICE > Bloomberg)
Anders Claesson (ICE)
Unnar Erlendsson (ICE > Google)
Tomas Magnusson (ICE > Google)
Émile Nadeau (ICE)
Jay Pantone (US)
Henning Ulfarsson (ICE)

CHRISTIAN BEAN, REYKJAVIK UNIVERSITY > UNIVERSITÉ PARIS 13

COMBINATORIAL EXPLORATION: GUIDED BY HUMANS, PROVEN BY COMPUTER

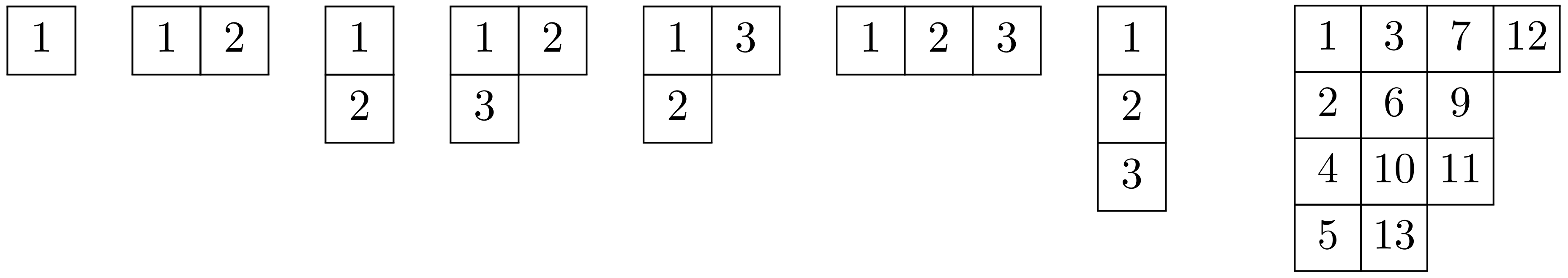
LIPN, UNIVERSITÉ PARIS 13, SÉMINAIRE DE COMBINATOIRE, DECEMBER 3, 2019

COMBINATORIAL OBJECTS

Permutations: A reordering of the integers $12\cdots n$.
25314 is a permutation of length 5

Words: Given an alphabet, say $\{a, b, c\}$, we consider all finite strings, such as $bbacbaac$

Standard Young Tableaux: Increasing rows, increasing columns



A **combinatorial set** is a set of objects with an associated size function such that there are finitely many objects of each size

ENUMERATIVE COMBINATORICS

FindStat: www.findstat.org
Database for information on and connections between combinatorial objects

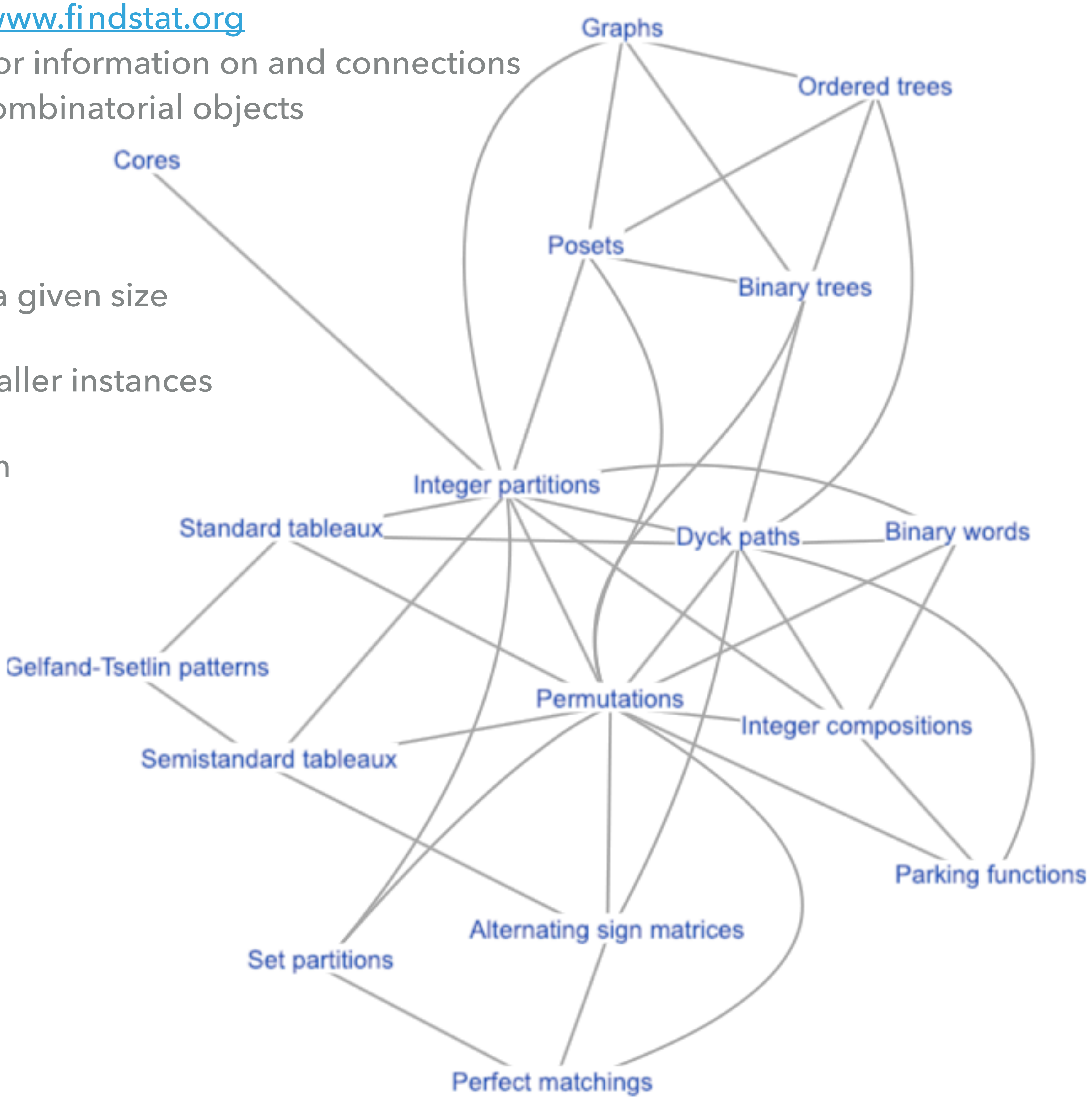
Counting how many objects are of a given size

Generate larger objects from smaller instances

Sample objects uniformly at random

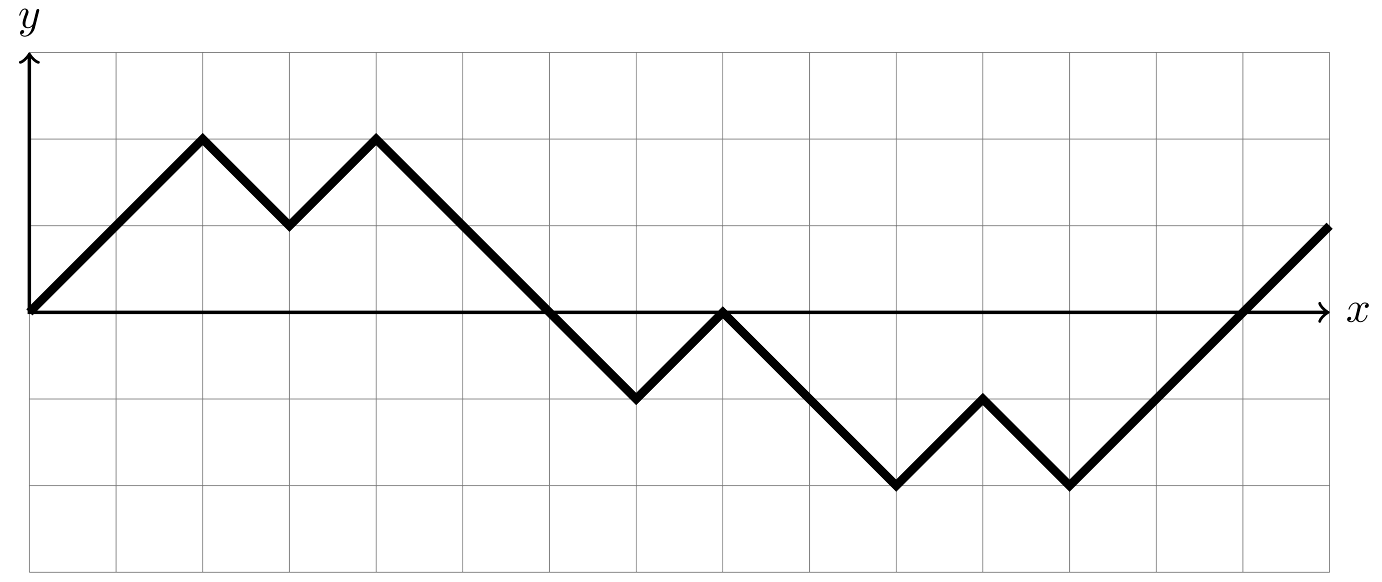
Find **bijections** to other objects

To answer these we need to understand the **structure** of the objects



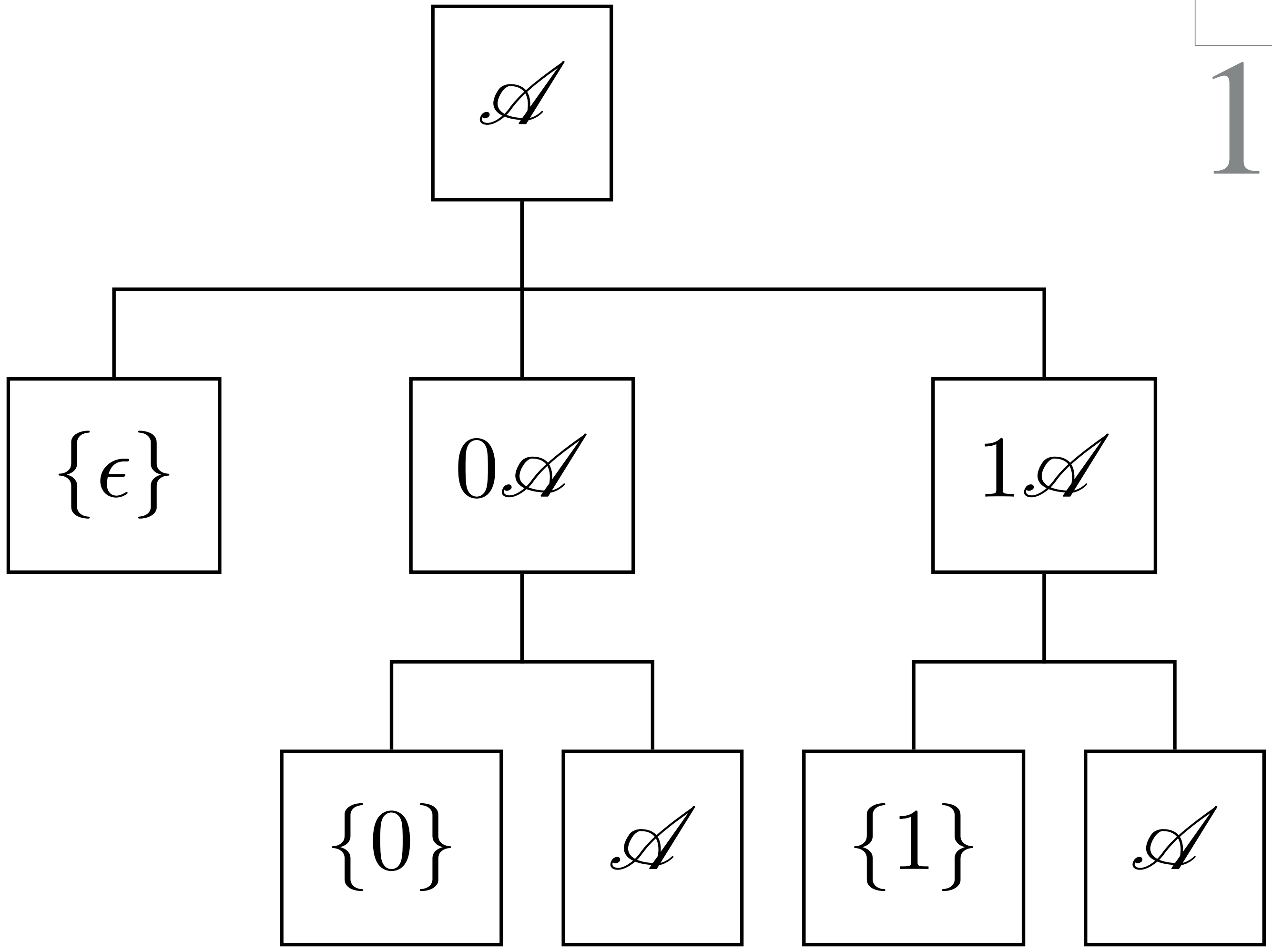
WALKING IN THE PLANE

We start with a simple example: **Walks in the plane** consisting of NE, and SE steps



110100010010111

We can build a path by appending either 0 or 1 to a shorter one



Generating functions: Quick introduction

WALKING IN THE PLANE

If we are interested in a **counting sequence** $(a_0, a_1, a_2, a_3, \ldots, a_n, \ldots)$
we can "store" it in the formal power series

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n + \cdots$$

This is the **generating function** of the counting sequence

The **constant sequence** $(1,1,1,1,1,1,\ldots)$ has the generating function

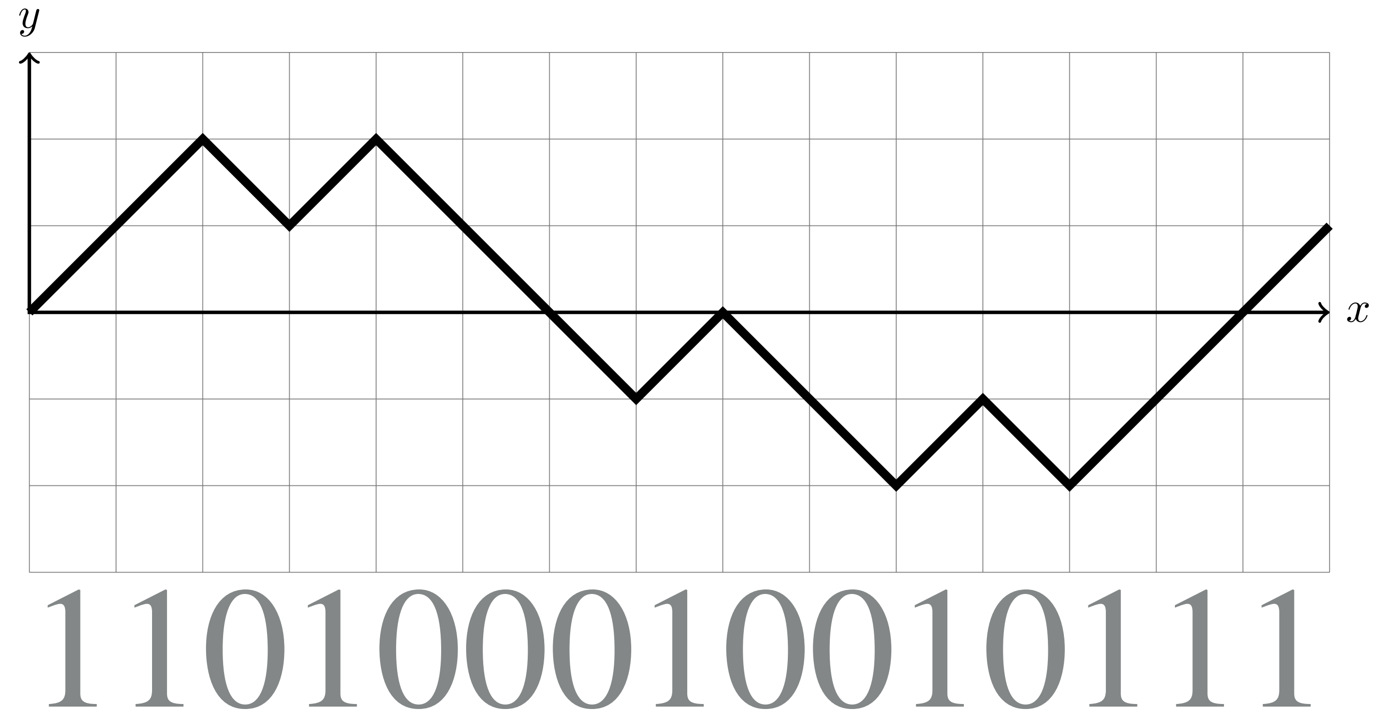
$$1 + x + x^2 + x^3 + x^4 + x^5 + \cdots = \frac{1}{1-x}$$

The **Fibonacci numbers** $(1,1,2,3,5,8,\ldots)$ have the generating function

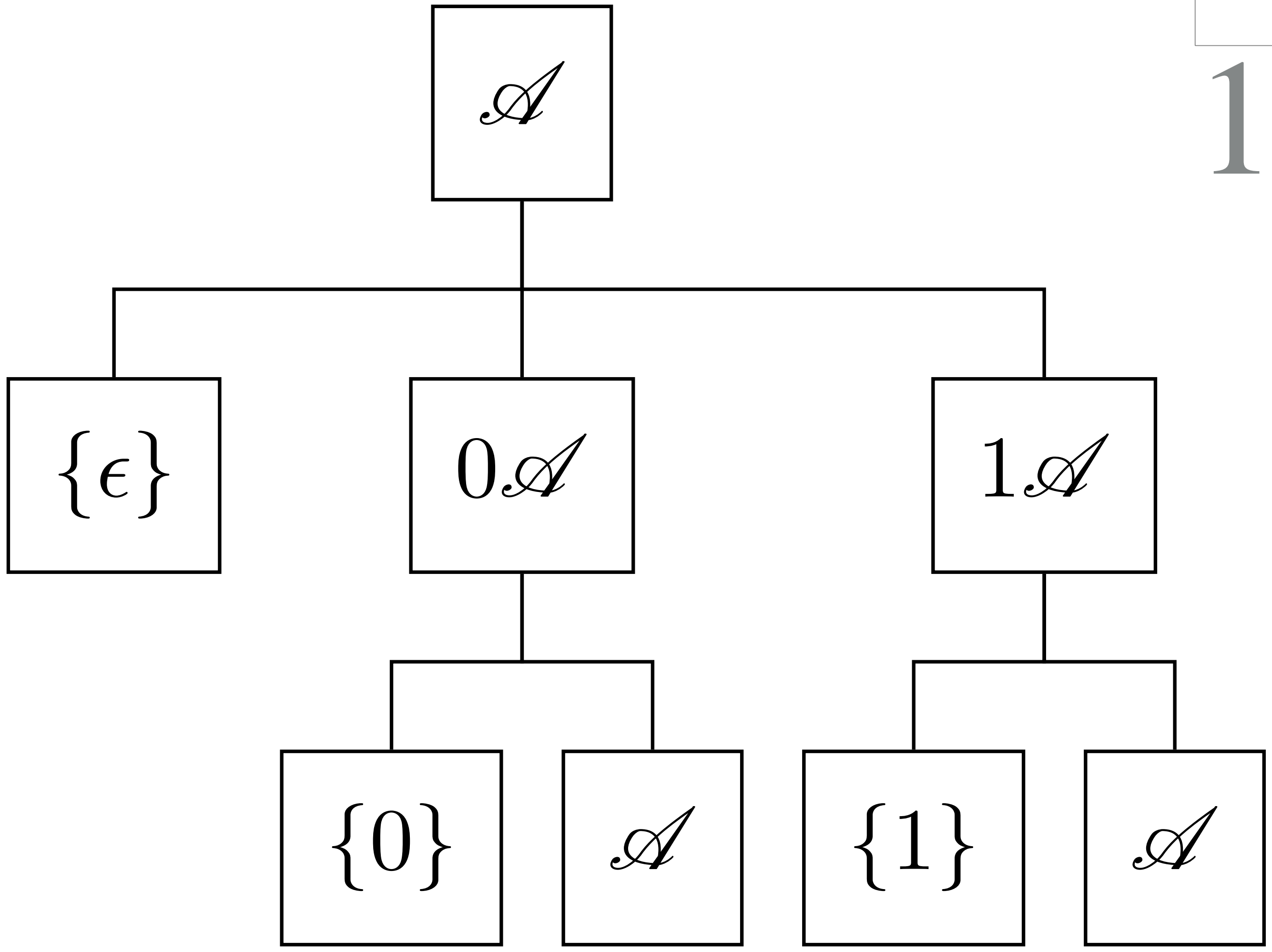
$$1 + x + 2x^2 + 3x^3 + 5x^4 + 8x^5 + \cdots = \frac{1}{1-x-x^2}$$

WALKING IN THE PLANE

We start with a simple example: **Walks in the plane** consisting of NE, and SE steps



We can build a path by appending either 0 or 1 to a shorter one



Generating functions

$A(x) = 1 + xA(x) + xA(x)$

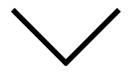
Solve to get

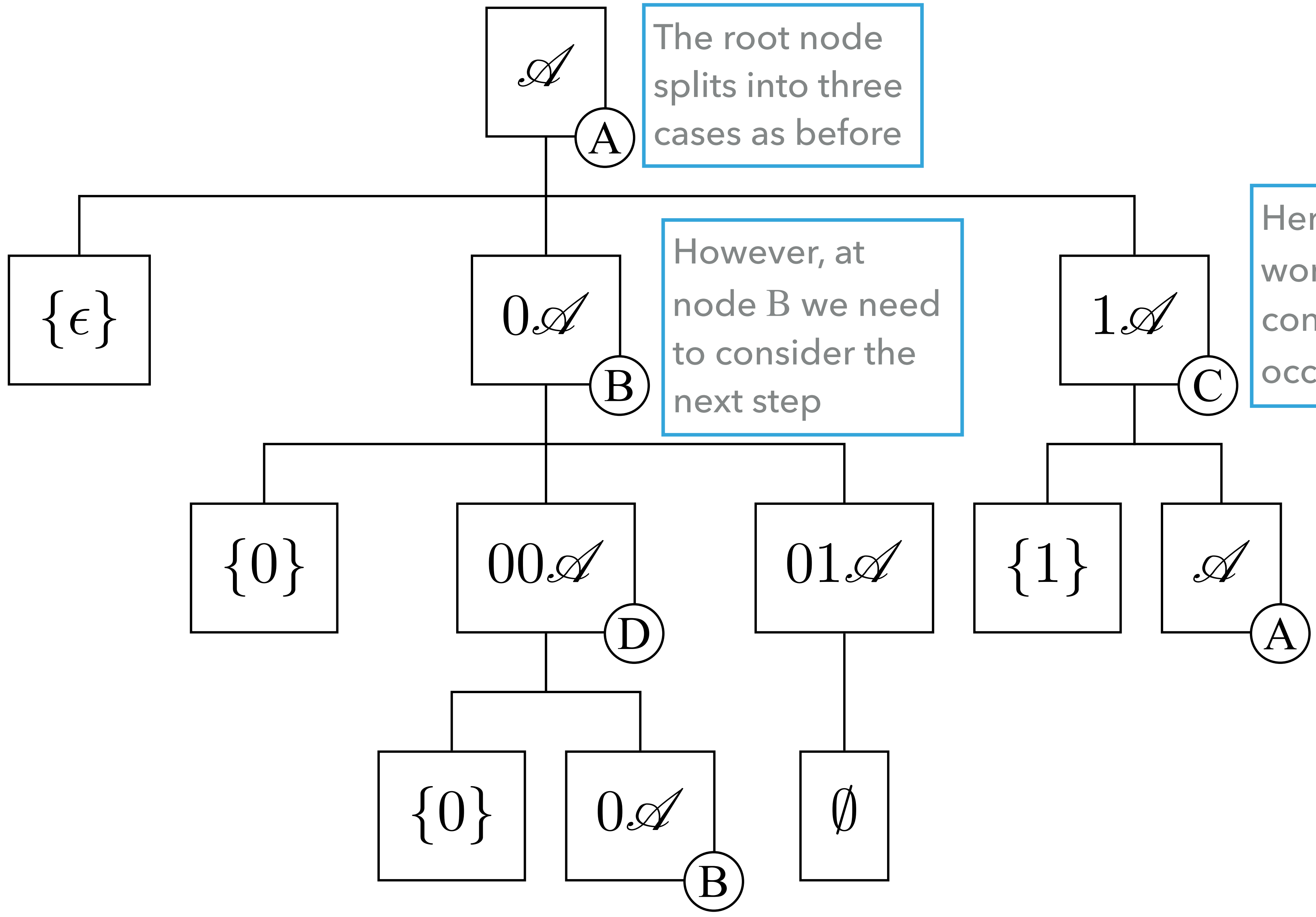
$$A(x) = \frac{1}{1-2x} = 1 + 2x + 4x^2 + 8x^3 + 16x^4 + \cdots$$

The structural description also allows us to generate objects, sample them, and perhaps find bijections

WALKING IN THE PLANE, WITH AVOIDANCE

We consider the same walks, but require they avoid consecutive 01





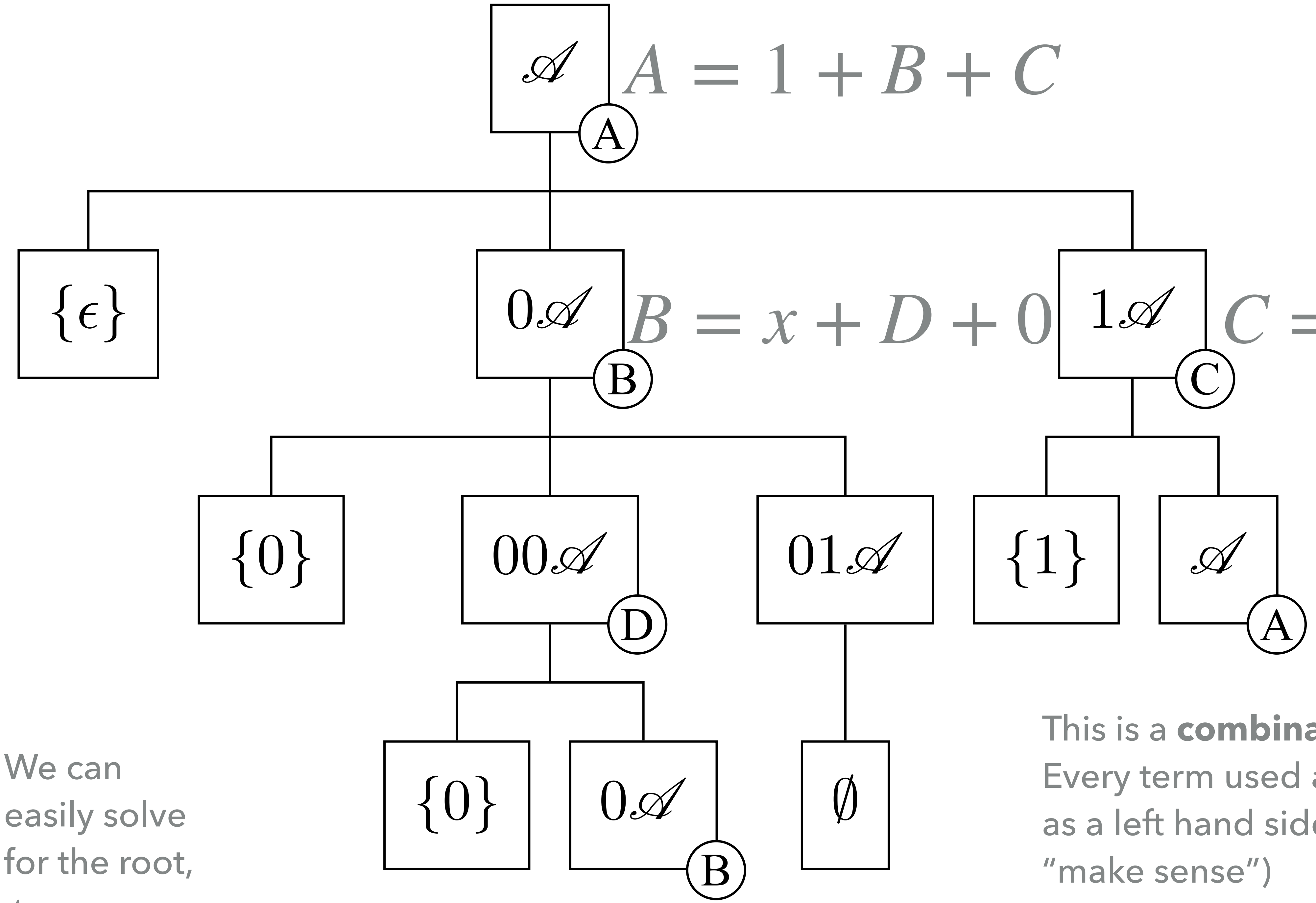
The root node splits into three cases as before

However, at node B we need to consider the next step

Here we do not need to worry about the initial 1 contributing to an occurrence of 01

We now move to the symbolic world

WALKING IN THE PLANE, CROSSING TO SYMBOLICS



$$A = 1 + B + C$$

$$B = x + D + 0$$

$$C = x \cdot A$$

$$D = x \cdot B$$

This is a **combinatorial specification**: Every term used appears exactly once as a left hand side (and the operations "make sense")

We can easily solve for the root, A, to get

$$A(x) = \frac{1}{(1-x)^2} = 1 + 2x + 3x^2 + 4x^3 + \cdots$$

Analytic Combinatorics: Treat the generating function as a complex function

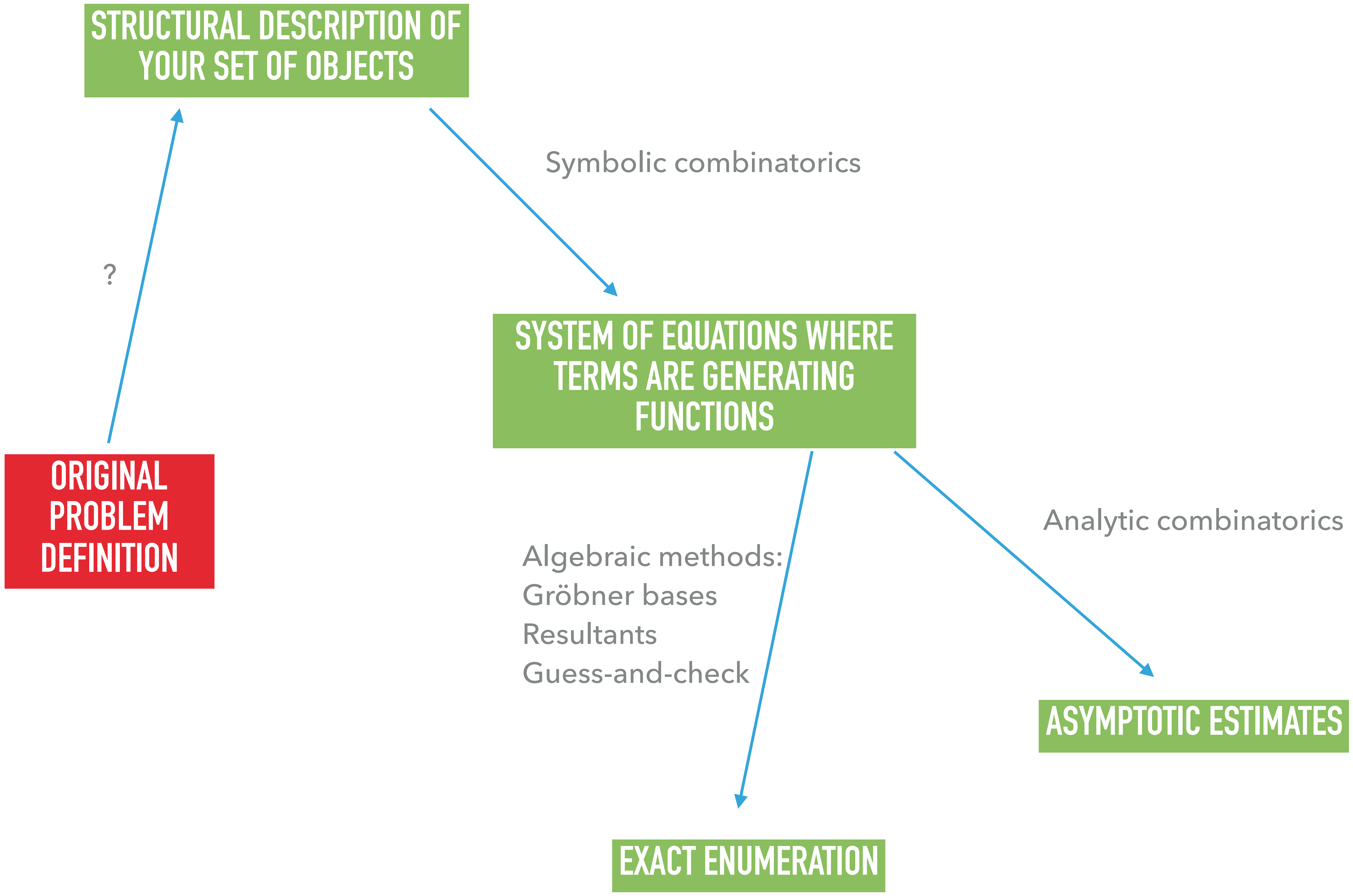
The Fibonacci numbers

$$\frac{1}{1-x-x^2} = 1 + x + 2x^2 + 3x^3 + \cdots$$

Have a pole at $x = \frac{\sqrt{5}-1}{2}$ giving us the exponential growth rate

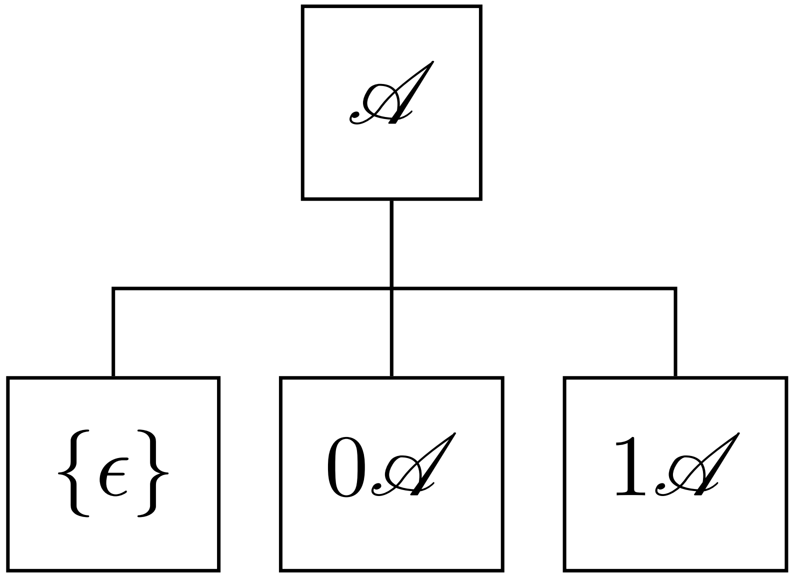
$$\left(\frac{2}{\sqrt{5}-1}\right)^n = \left(\frac{\sqrt{5}+1}{2}\right)^n$$

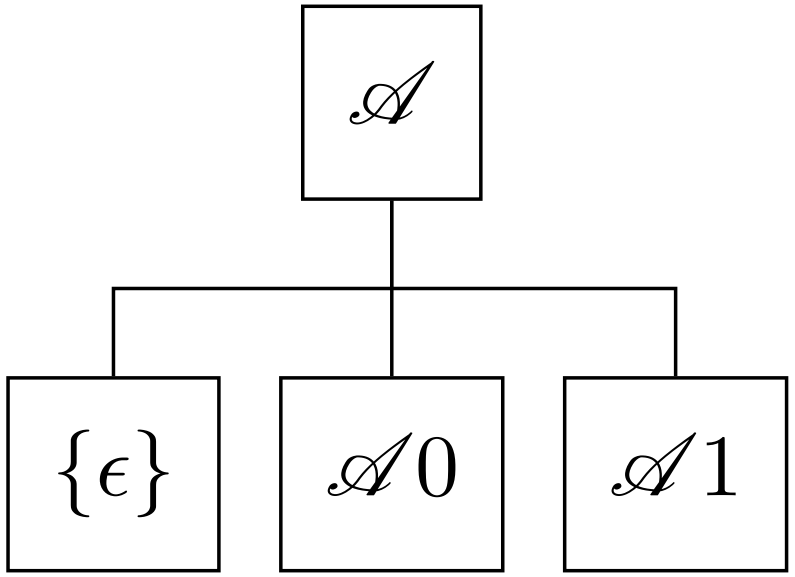
THE PIPELINE, AND THE GAP



AN ATTEMPT TO FILL THE GAP: COMBINATORIAL EXPLORATION

Why did we consider the first letter in the walks?





Why not consider both, and build a **universe** of relations between combinatorial objects?

That's what we do: Given **strategies** written by the user, **apply them in every possible way** to all available objects and build a massive universe of relations. So far we have only seen one type of strategy: Batch strategies, which break an object into cases, or factors

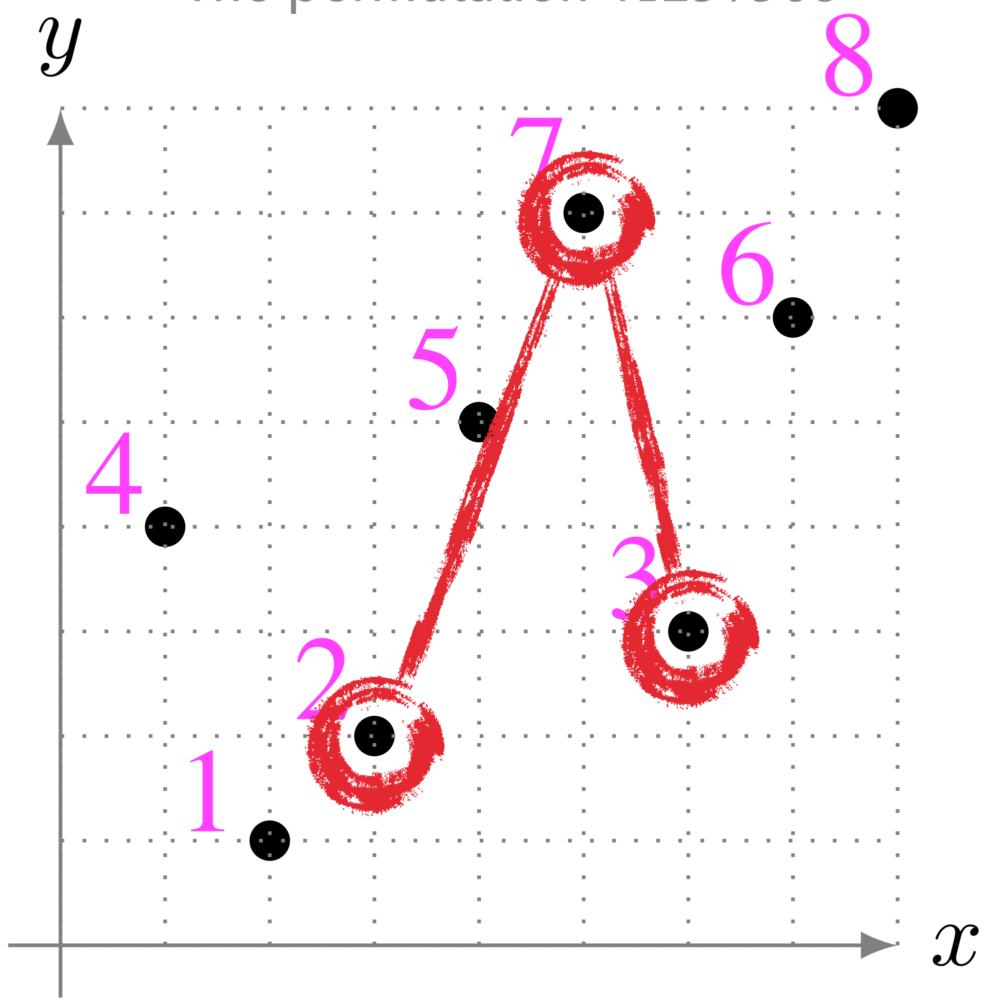
For the domain of permutation patterns, we'll see other types of strategies

THE ROOTS OF PATTERN AVOIDING PERMUTATIONS

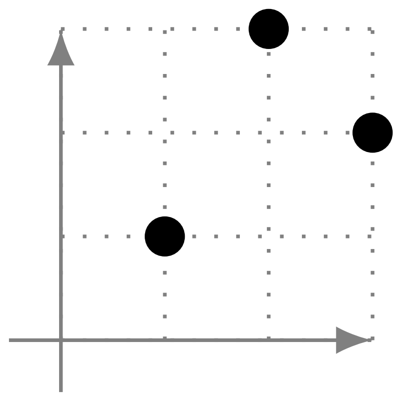
In an exercise in Chapter 2 of Knuth's *The Art of Computer Programming* we are asked to enumerate permutations that **avoid** 132.

In this definition of avoidance we do **not** require entries to be adjacent

The permutation 41257368



This permutation does not avoid 132. It contains many copies of 132. For example in the subsequence 273.



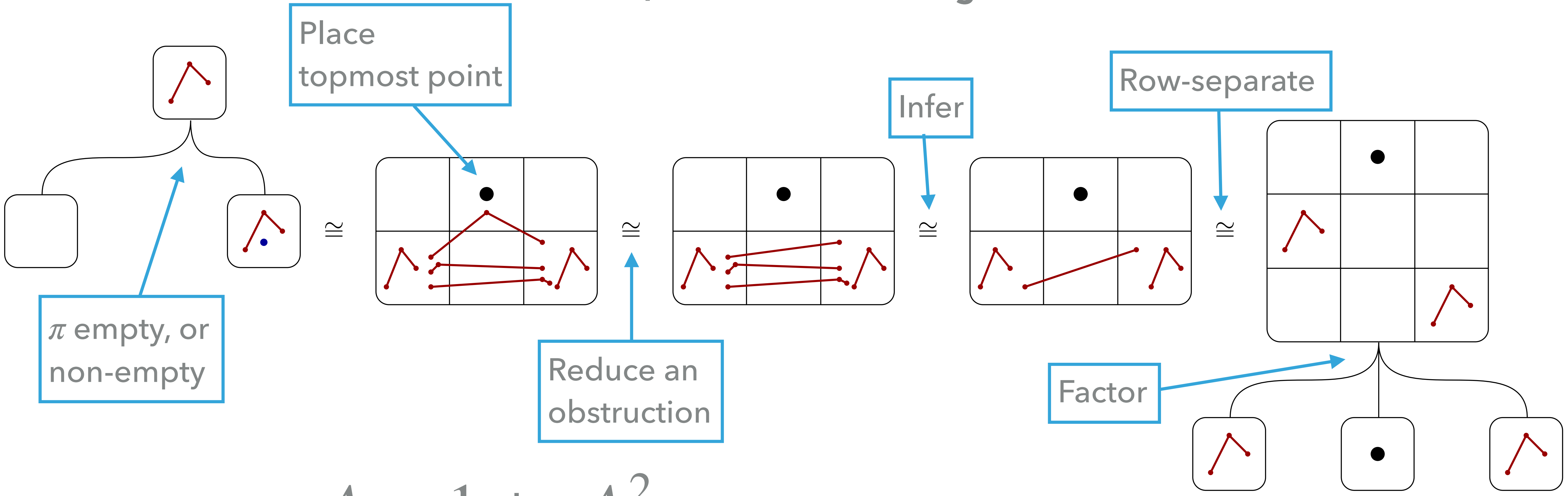
ANSWERING KNUTH'S QUESTION

Usually, this is done as follows: Consider a non-empty permutation π that avoids 132. The maximum element of π, call it n, is somewhere

$$\pi = L\ n\ R$$

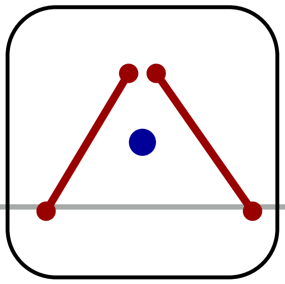
Of course L and R must avoid 132. In addition, every entry of L should be greater than every entry of R.

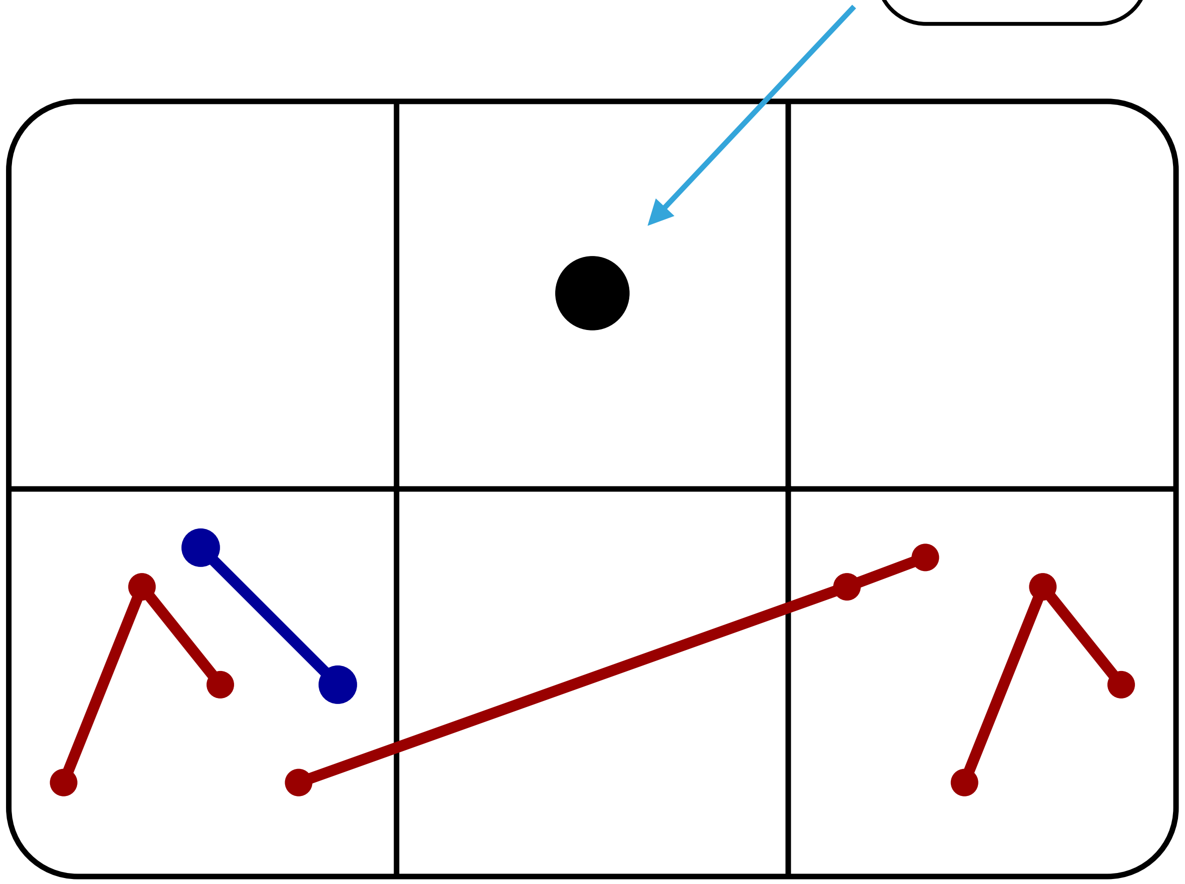
Now, without handwaving



$$A = 1 + xA^2$$

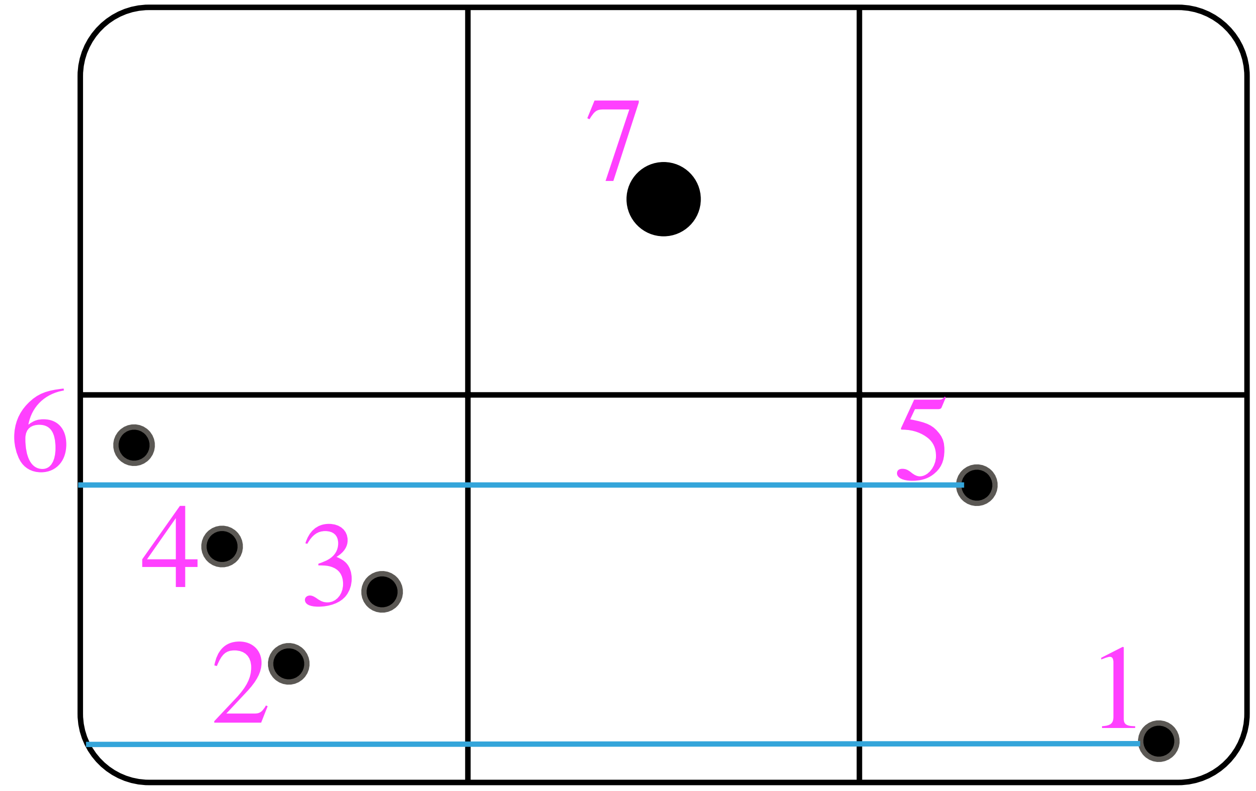
TILINGS





A *tiling* is a triple $((n, m), \mathcal{O}, \mathcal{R})$, where (n, m) are the *dimensions*, $\mathcal{O}$ are the *obstructions*, and $\mathcal{R}$ are the *requirements*

The tiling *represents* the set of (gridded) permutations that can be drawn on the tiling, without containing **any** obstruction, while containing **every** requirement

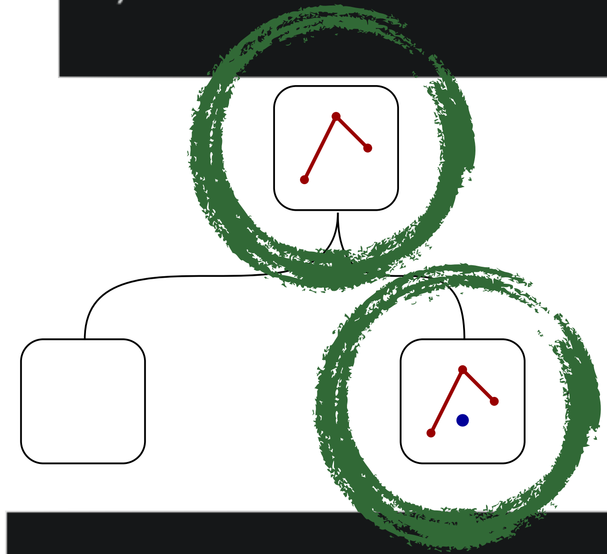


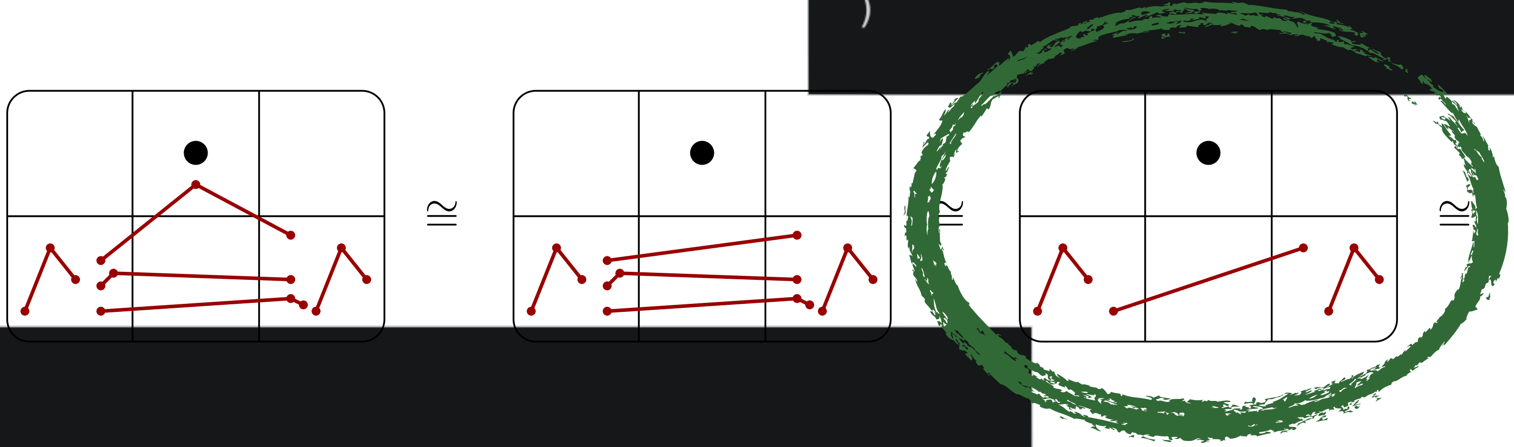
Here we get the permutation 6423751, although, strictly speaking we should also write the coordinate of each point

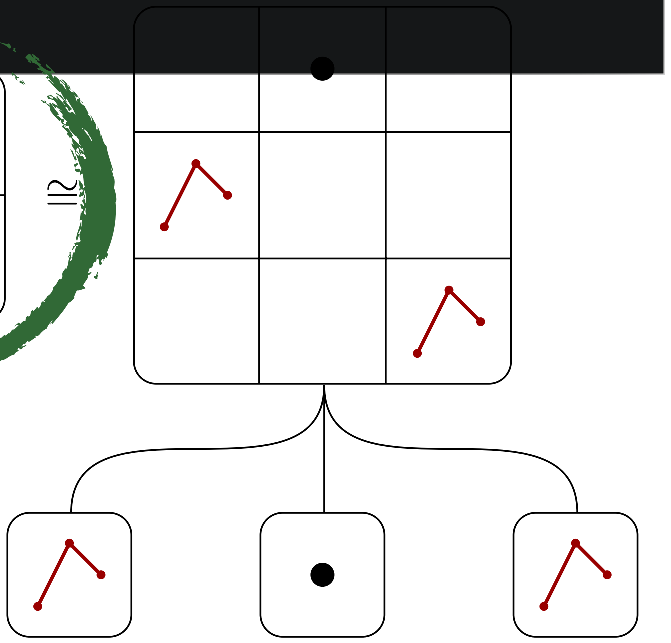
MOVING TO THE COMPUTER

```
Tiling(
  obstructions=(
    Obstruction(Perm(0,2,1), ((0,0), (0,0), (0,0)))
  ),
  requirements=()
)
```

```
Tiling(
  obstructions=(
    Obstruction(Perm(0), ((0,1),)),
    Obstruction(Perm(0), ((1,0),)),
    Obstruction(Perm(0), ((2,1),)),
    Obstruction(Perm(0,1), ((0,0), (2,0))),
    Obstruction(Perm(0,1), ((1,1), (1,1))),
    Obstruction(Perm(1,0), ((1,1), (1,1))),
    Obstruction(Perm(0,2,1), ((0,0), (0,0), (0,0))),
    Obstruction(Perm(0,2,1), ((2,0), (2,0), (2,0)))
  ),
  requirements=(
    (Requirement(Perm(0), ((1,1))))
                )
)
```







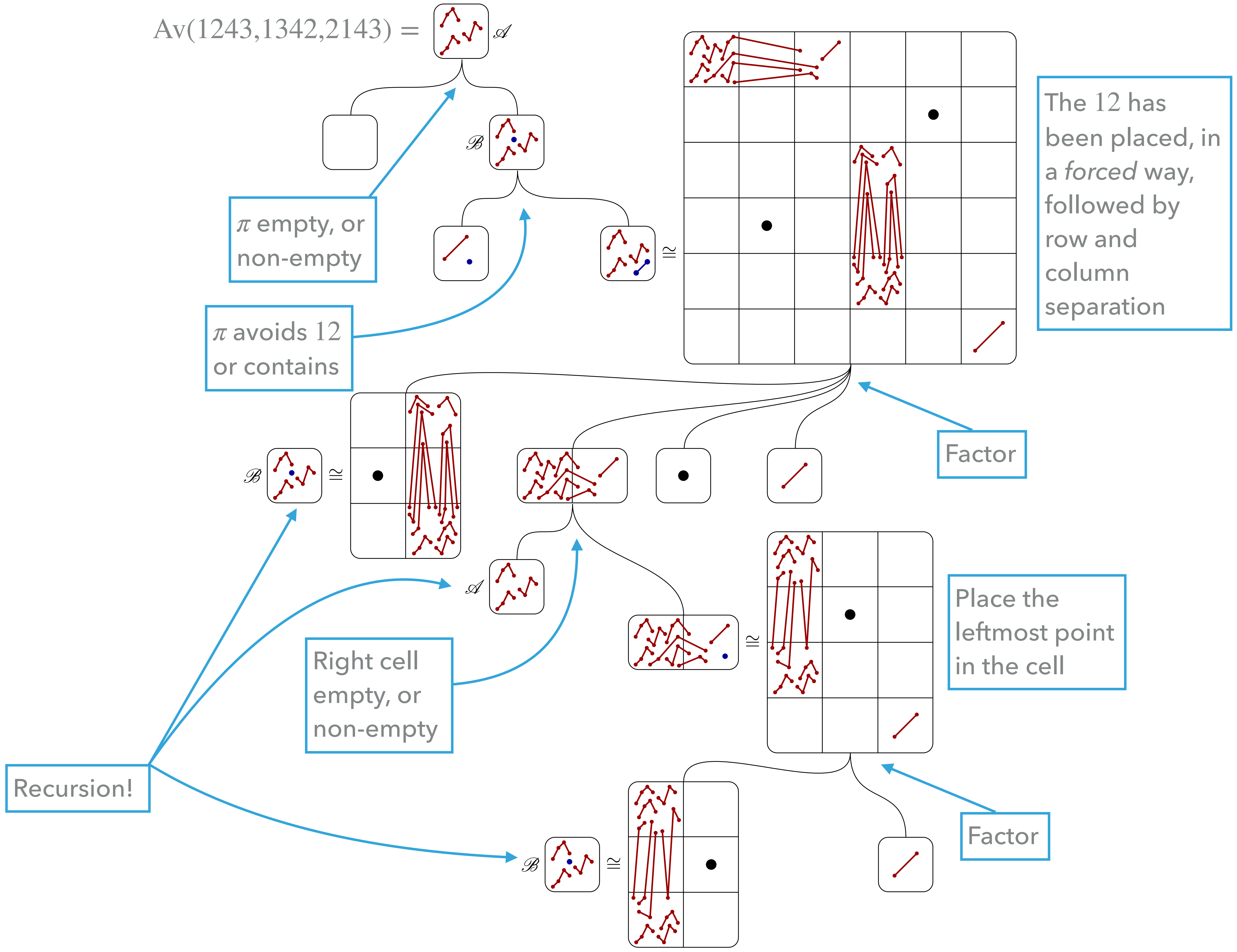
```
Tiling(
  obstructions=(
    Obstruction(Perm(0,2,1), ((0,0), (0,0), (0,0)))
  ),
  requirements=(
    (Requirement(Perm(0), ((0,0))))
  )
)
```

LET´S DO IT!

```
combrunner 132 point_placements
```

```
Initialising CombSpecSearcher for the combinatorial class:
+-+
|1|
+-+
1: Av(021)
Crossing obstructions:

Looking for recursive combinatorial specification with the strategies:
Inferral: row_and_column_separation, obstruction_transitivity
Initial: factor, requirement_corroboration
Verification: subset_verified, globally_verified
Set 1: all_cell_insertions
Set 2: requirement_placement
```



$\mathrm{Av}(1243,1342,2143) =$ $\mathscr{A}$
$\mathscr{B}$
$\cong$
π empty, or non-empty
π avoids 12 or contains
The 12 has been placed, in a *forced* way, followed by row and column separation
Factor
$\mathscr{B}$ $\cong$
$\mathscr{A}$
Right cell empty, or non-empty
Place the leftmost point in the cell
Factor
Recursion!
$\mathscr{B}$ $\cong$

WHY PATTERN AVOIDING PERMUTATIONS?

Short answer: My PhD supervisor told me to

Real answer: They are equivalent to many other objects in discrete mathematics

Smooth Schubert varieties (type A) = $\mathrm{Av}(3412,4231)$

Stack sortable permutations (Knuth) = $\mathrm{Av}(132)$

Rooted non-separable planar maps = permutations avoiding generalized patterns

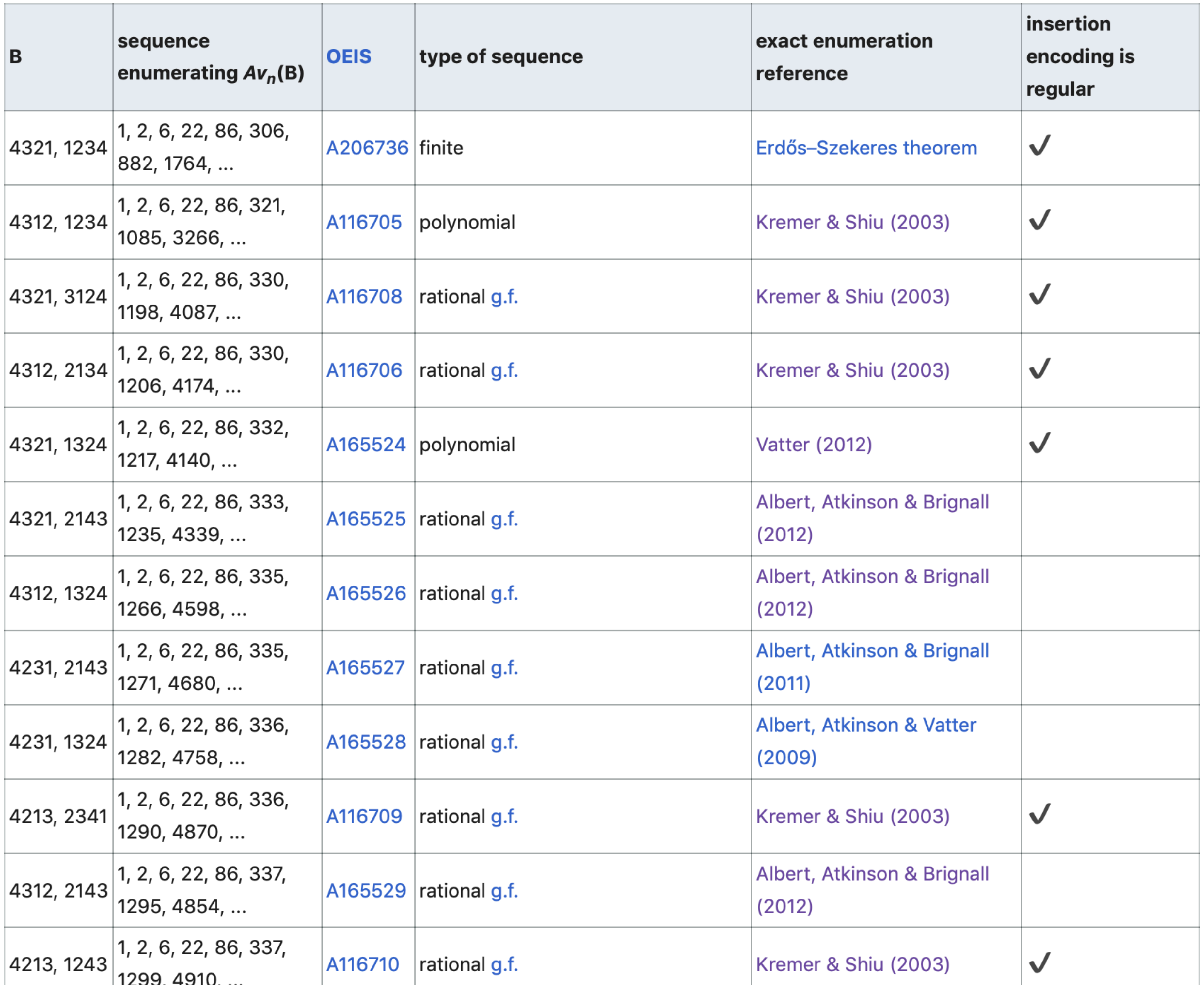
There is a sharp jump in difficulty from the problem we considered ($\mathrm{Av}(132)$) to avoiding longer patterns, for example no one knows how to count $\mathrm{Av}(1324)$, and Zeilberger even claimed that not even God knows how many permutations of length 1000 avoid 1324.

So in recent decades the focus has been on enumerating permutations avoiding several patterns of length 4.

COMBINATORIAL EXPLORATION: SUCCESSES

The testing ground for new approaches has been the set of permutations avoiding two length 4 patterns. A total of 56 problems

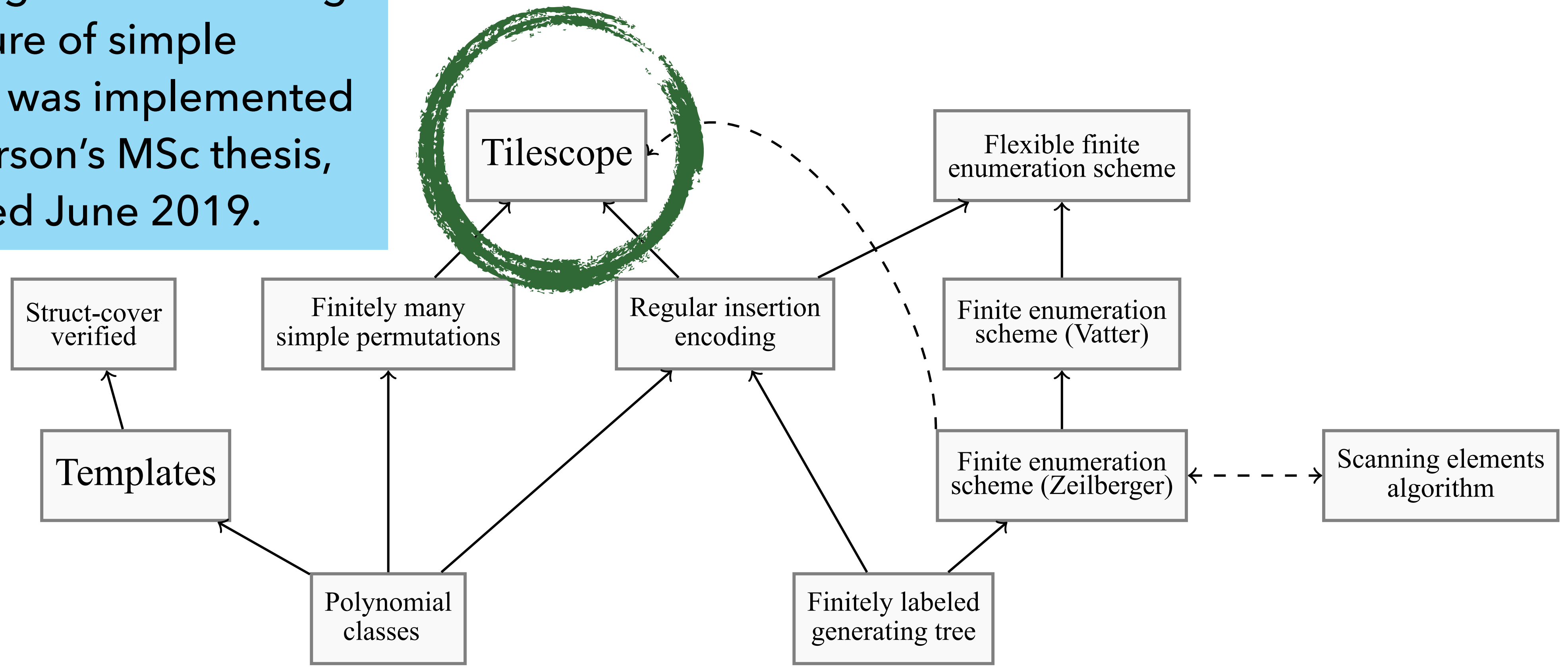
Wikipedia page: Enumerations of specific permutation classes



B	sequence enumerating Av_n(B)	OEIS	type of sequence	exact enumeration reference	insertion encoding is regular
4321, 1234	1, 2, 6, 22, 86, 306, 882, 1764, ...	A206736	finite	Erdős–Szekeres theorem	✓
4312, 1234	1, 2, 6, 22, 86, 321, 1085, 3266, ...	A116705	polynomial	Kremer & Shiu (2003)	✓
4321, 3124	1, 2, 6, 22, 86, 330, 1198, 4087, ...	A116708	rational g.f.	Kremer & Shiu (2003)	✓
4312, 2134	1, 2, 6, 22, 86, 330, 1206, 4174, ...	A116706	rational g.f.	Kremer & Shiu (2003)	✓
4321, 1324	1, 2, 6, 22, 86, 332, 1217, 4140, ...	A165524	polynomial	Vatter (2012)	✓
4321, 2143	1, 2, 6, 22, 86, 333, 1235, 4339, ...	A165525	rational g.f.	Albert, Atkinson & Brignall (2012)	
4312, 1324	1, 2, 6, 22, 86, 335, 1266, 4598, ...	A165526	rational g.f.	Albert, Atkinson & Brignall (2012)	
4231, 2143	1, 2, 6, 22, 86, 335, 1271, 4680, ...	A165527	rational g.f.	Albert, Atkinson & Brignall (2011)	
4231, 1324	1, 2, 6, 22, 86, 336, 1282, 4758, ...	A165528	rational g.f.	Albert, Atkinson & Vatter (2009)	
4213, 2341	1, 2, 6, 22, 86, 336, 1290, 4870, ...	A116709	rational g.f.	Kremer & Shiu (2003)	✓
4312, 2143	1, 2, 6, 22, 86, 337, 1295, 4854, ...	A165529	rational g.f.	Albert, Atkinson & Brignall (2012)	
4213, 1243	1, 2, 6, 22, 86, 337, 1299, 4910, ...	A116710	rational g.f.	Kremer & Shiu (2003)	✓

COMBINATORIAL EXPLORATION: SUCCESSES

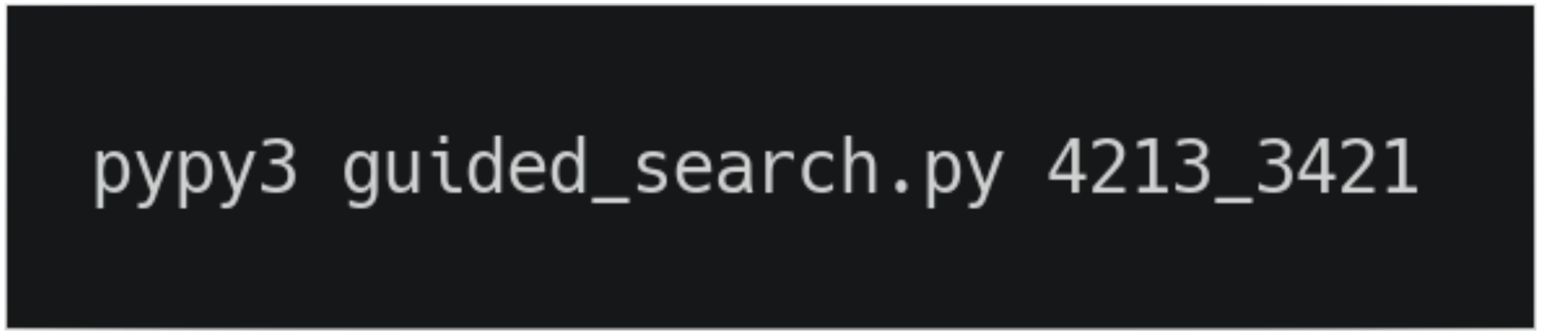
A set of strategies for searching for the structure of simple permutations was implemented in Arnar Arnarson's MSc thesis, who graduated June 2019.


Tilescope
Flexible finite enumeration scheme
Struct-cover verified
Finitely many simple permutations
Regular insertion encoding
Finite enumeration scheme (Vatter)
Templates
Finite enumeration scheme (Zeilberger)
Scanning elements algorithm
Polynomial classes
Finitely labeled generating tree

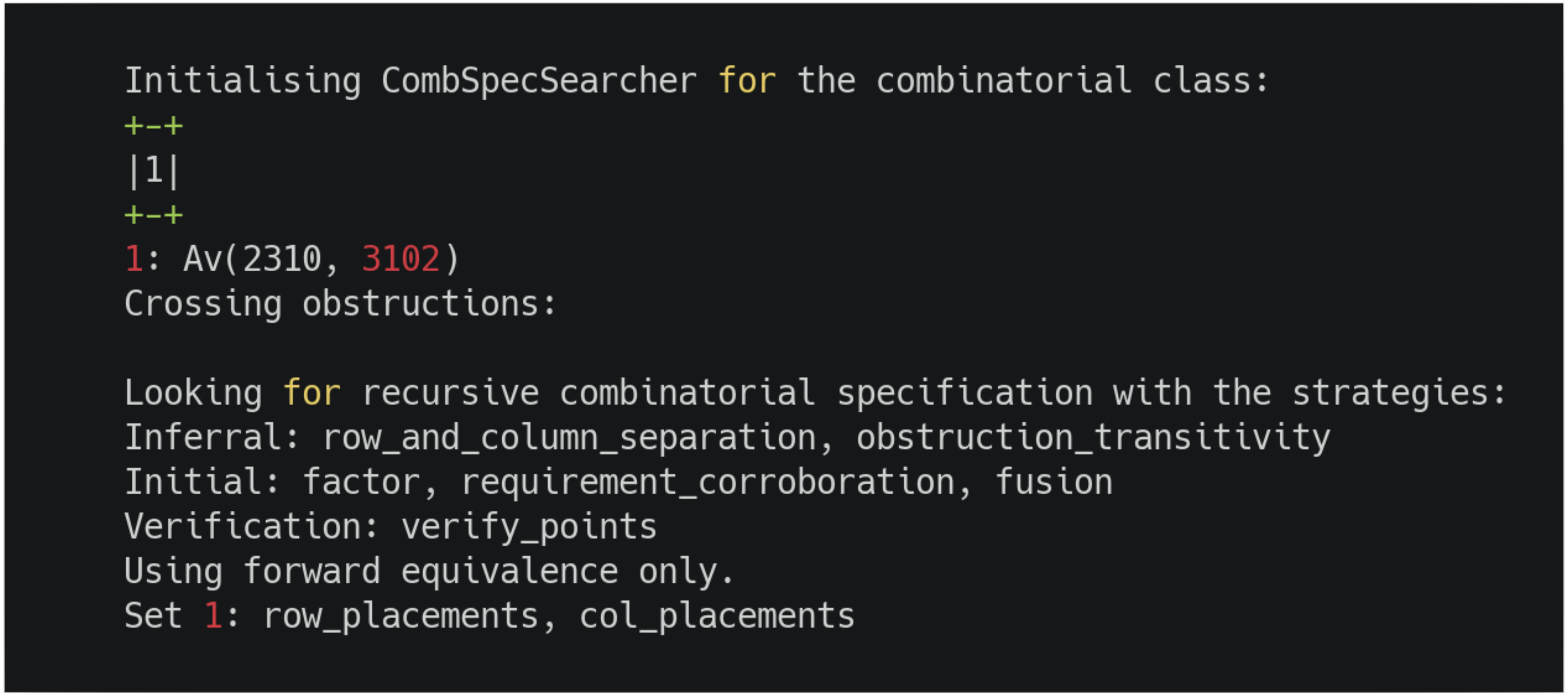
About a month ago we were finally able to complete the last of the 56 problems, by finding a tree for Av(1432,2143), which had 888 nodes and took 4 days on a very powerful computer. That class was done in 2018 by humans but has not been published, besides on the arXiv

For every one of these successes we can write down a system of equations, sometimes in several variables. We have always been able to solve when there is a single variable, but when there are more we can turn the solution into a polynomial time algorithm for the counting sequence

Av(4213,3421) FIRST ENUMERATED BY BEVAN IN 2016

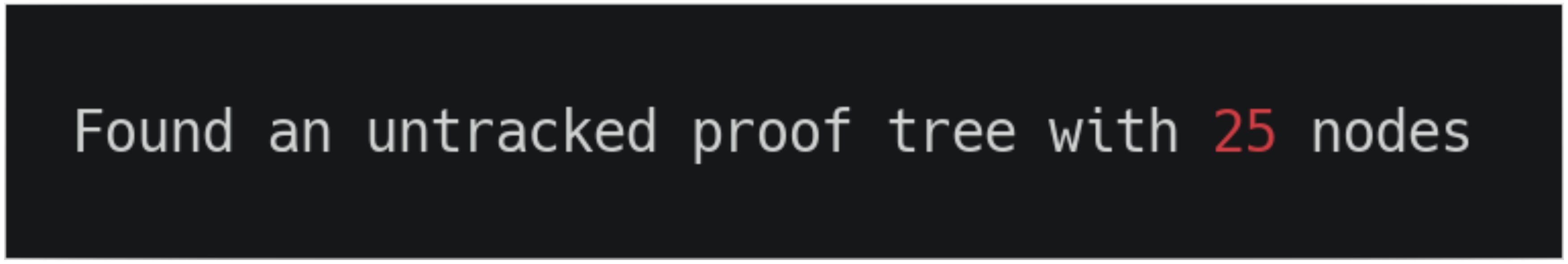


```
pypy3 guided_search.py 4213_3421
```



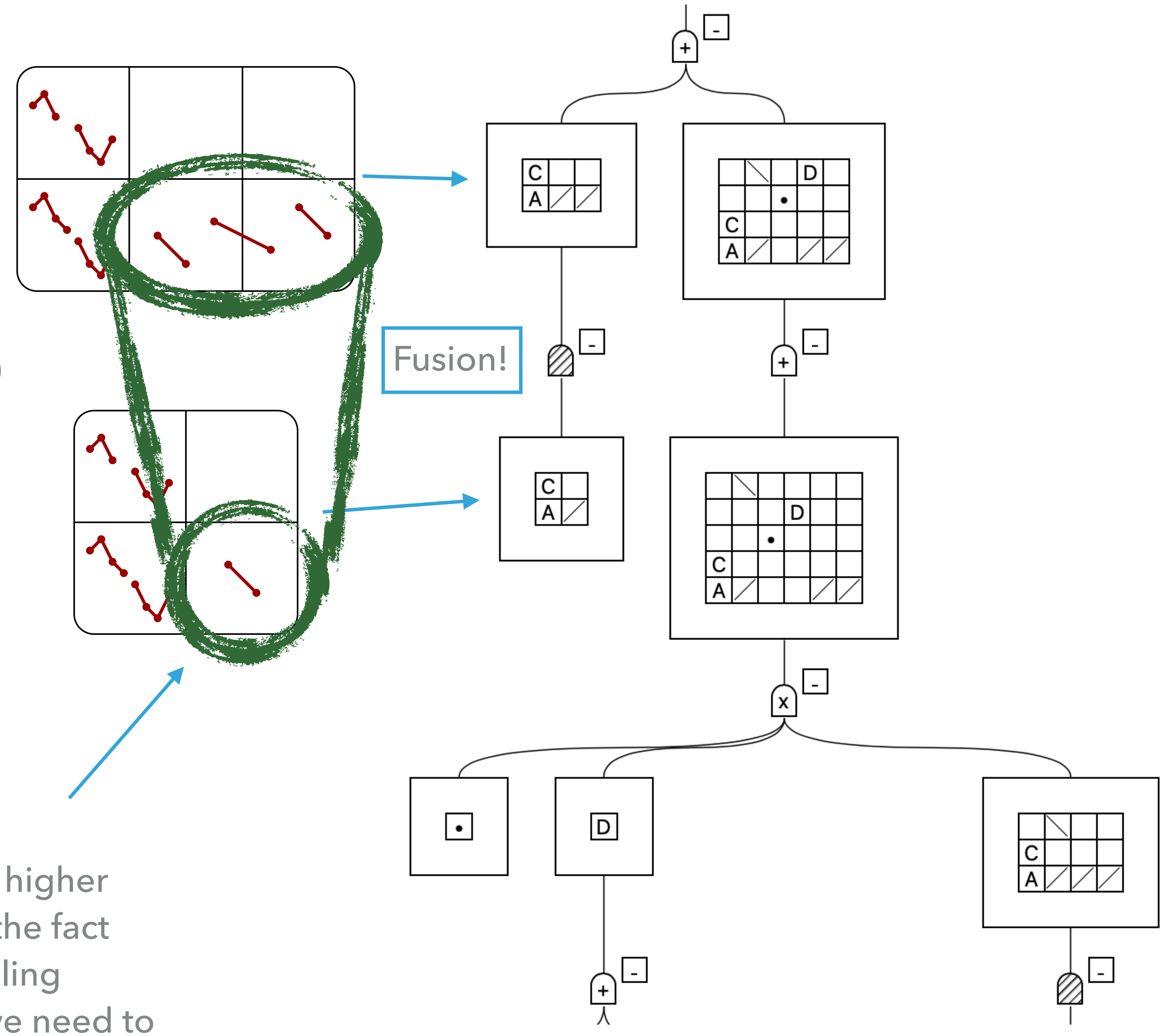
```
Initialising CombSpecSearcher for the combinatorial class:
+-+
|1|
+-+
1: Av(2310, 3102)
Crossing obstructions:

Looking for recursive combinatorial specification with the strategies:
Inferral: row_and_column_separation, obstruction_transitivity
Initial: factor, requirement_corroboration, fusion
Verification: verify_points
Using forward equivalence only.
Set 1: row_placements, col_placements
```



```
Found an untracked proof tree with 25 nodes
```

FUSION TREE

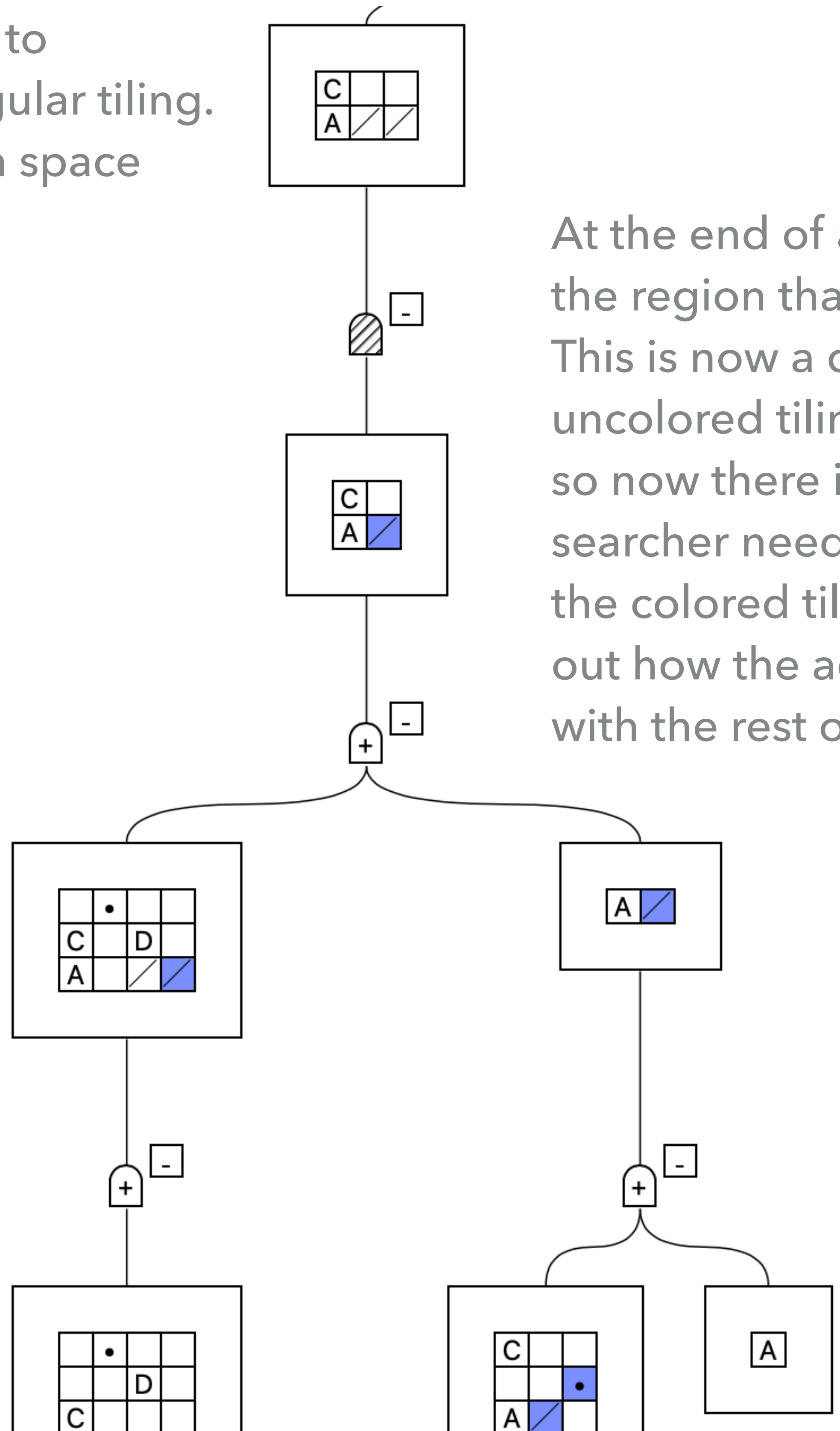


(Not all obstructions drawn on these tilings)

In the tree the fused tiling recurses to a node higher in the tree. This ignores the fact that to count the fused tiling using the unfused one we need to know the number of points in the decreasing run

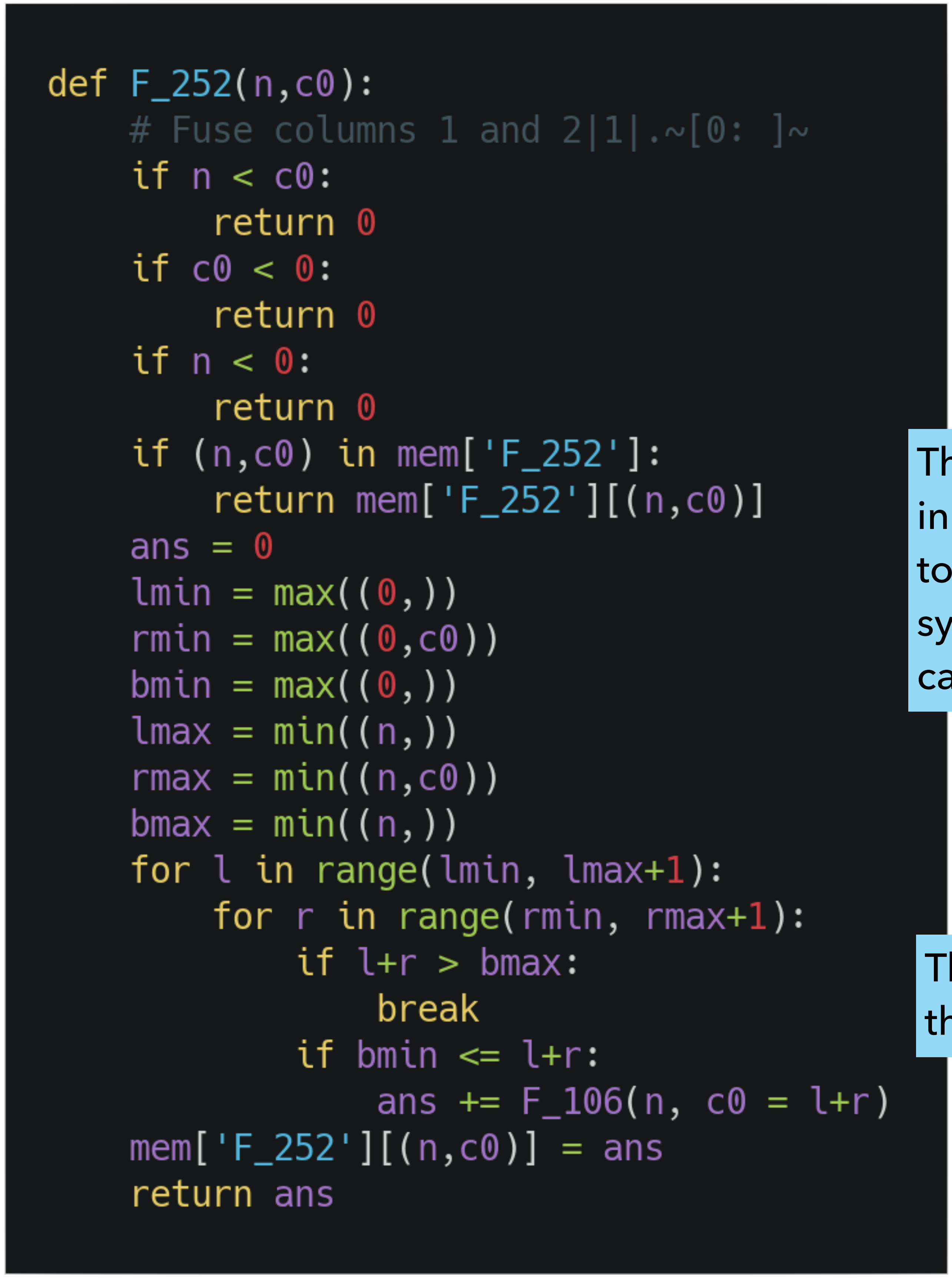
TRACKED TREE

We start a new search, that restricts the space to "colored tilings" with the same underlying regular tiling. This is done to limit an explosion in the search space

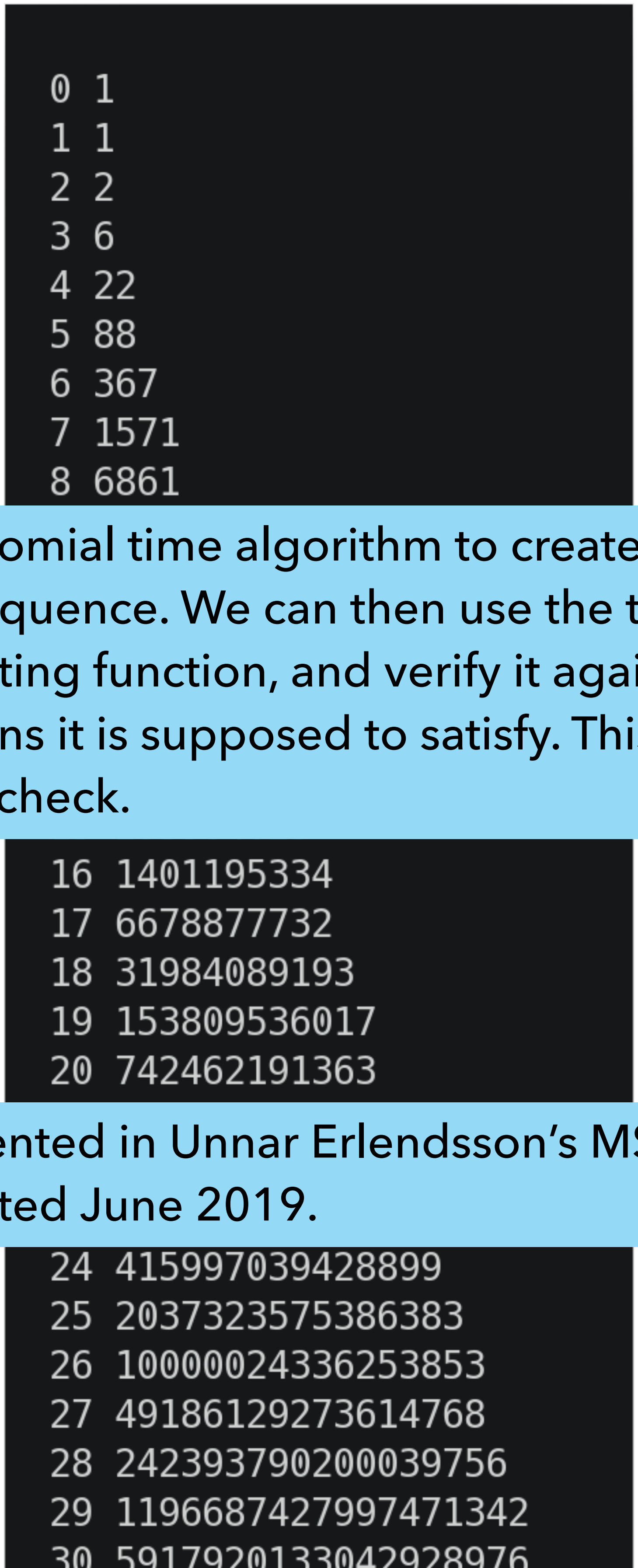


At the end of a fusion step we color the region that needs to be tracked. This is now a different node than the uncolored tiling higher in the tree, so now there is no recursion, and the searcher needs to keep working on the colored tiling, eventually working out how the added color interacts with the rest of the tree

TURNING THE TREE INTO RECURRENCE RELATIONS



```python
def F_252(n,c0):
    # Fuse columns 1 and 2|1|.~[0: ]~
    if n < c0:
        return 0
    if c0 < 0:
        return 0
    if n < 0:
        return 0
    if (n,c0) in mem['F_252']:
        return mem['F_252'][(n,c0)]
    ans = 0
    lmin = max((0,))
    rmin = max((0,c0))
    bmin = max((0,))
    lmax = min((n,))
    rmax = min((n,c0))
    bmax = min((n,))
    for l in range(lmin, lmax+1):
        for r in range(rmin, rmax+1):
            if l+r > bmax:
                break
            if bmin <= l+r:
                ans += F_106(n, c0 = l+r)
    mem['F_252'][(n,c0)] = ans
    return ans
```



```
0 1
1 1
2 2
3 6
4 22
5 88
6 367
7 1571
8 6861
```

This gives a polynomial time algorithm to create terms in the counting sequence. We can then use the terms to guess a generating function, and verify it against the system of equations it is supposed to satisfy. This is called guess-and-check.

```
16 1401195334
17 6678877732
18 31984089193
19 153809536017
20 742462191363
```

This was implemented in Unnar Erlendsson’s MSc thesis. He graduated June 2019.

```
24 415997039428899
25 2037323575386383
26 10000024336253853
27 49186129273614768
28 242393790200039756
29 1196687427997471342
30 5917920133042928976
```

COMBINATORIAL EXPLORATION: WRAPPING UP

We have implemented a framework, https://pypi.org/project/comb-spec-searcher/ in Python, which takes care of all the searching, and book-keeping. To use it for permutation patterns we implemented tilings https://pypi.org/project/tilings to encode our objects and strategies. These are both open source.

If you want to use the framework you need to implement a representation of your favourite combinatorial sets, and strategies to manipulate them. There is a readme at the CombSpecSearcher link above that goes through a basic example of how to do consecutive pattern avoidance in words.

The algorithm is guaranteed to find a combinatorial specification if it exists in the universe. As we'll see later, sometimes the universe contains the answer to your question without having any combinatorial specification.

(This assumes you have implemented sane strategies)

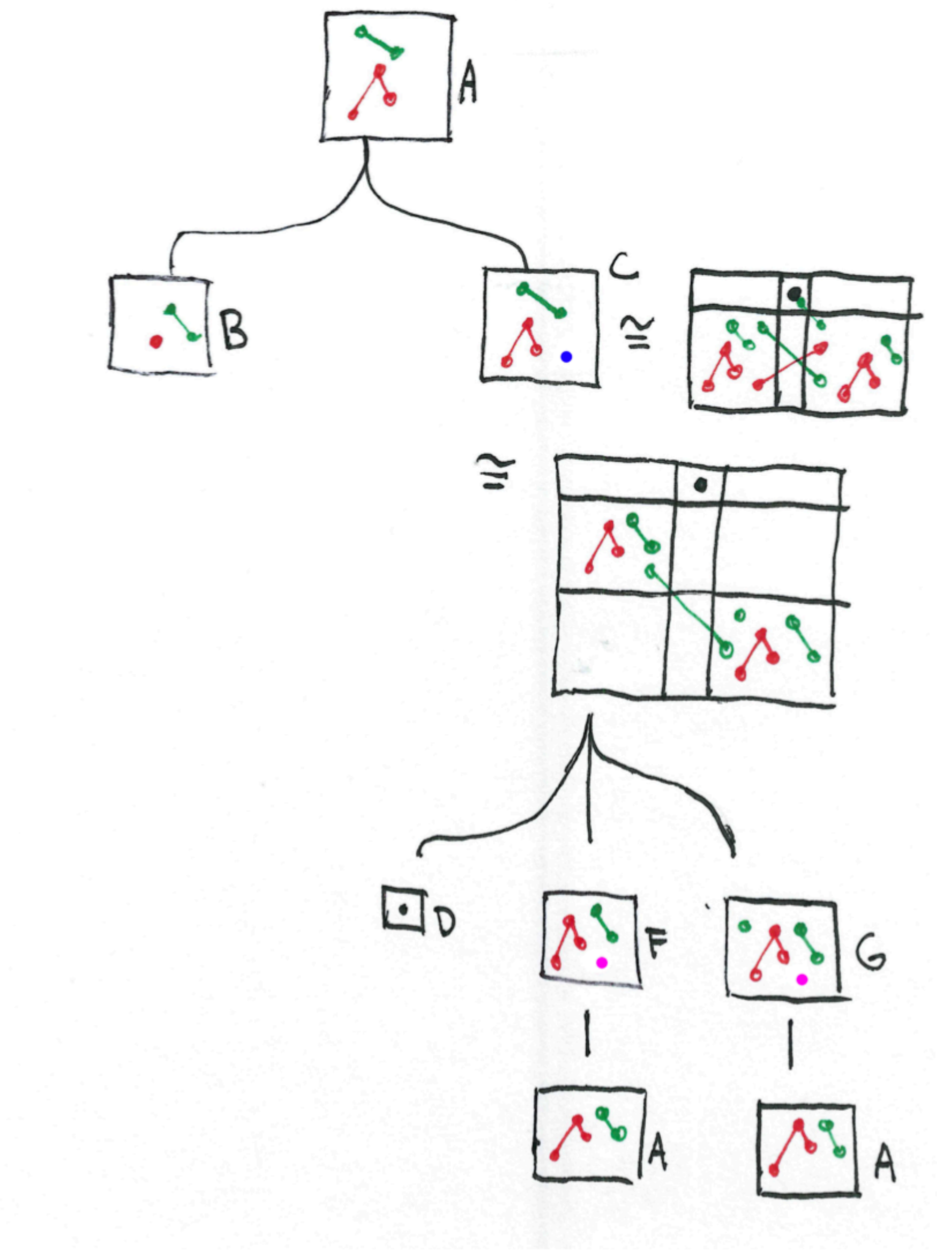
PRODUCTIVE STRATEGIES IMPLY PRODUCTIVE PROOF TREES

Definition 3. *We call a k-ary strategy S a* productive strategy *if the following conditions hold for all combinatorial sets $\mathcal{A}$ with corresponding decomposition $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(k)})$, and for all $i \in \{1, \ldots, k\}$. Once again, we use the phrase "$\mathcal{A}_n$ relies on $\mathcal{B}_j^{(i)}$" or the diagram $\mathcal{A}_n \to \mathcal{B}_j^{(i)}$ as a simplified way of stating the formal information contained in the reliance profile function of S, namely that $j \in (r_S(n))_i$.*

1. *If $\mathcal{A}_n$ relies on $\mathcal{B}_j^{(i)}$, then $n \geq j$.*
2. *$\mathcal{A}$ is not equinumerous to $\mathcal{B}^{(i)}$.*
3. *If $\mathcal{A}_n$ relies on $\mathcal{B}_n^{(i)}$, then $|\mathcal{A}_n| \geq |\mathcal{B}_n^{(i)}|$.*
4. *If $\mathcal{A}_n$ relies on $\mathcal{B}_n^{(i)}$ for some n, then $\mathcal{A}_N$ relies on $\mathcal{B}_N^{(i)}$ for all $N \in \mathbb{N}$.*

Theorem 3. *Let P be a proof tree, or the equivalent combinatorial specification, composed entirely of productive strategies. Then P is a productive proof tree, or equivalently a productive combinatorial specification.*

TRACKING STATISTICS?



A tree tracking inversions in Av(132)

$$A(n,k) = B(n,k) + C(n,k).$$

$$B(n,k) = \begin{cases} 1 & \text{if } n = k = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$C(n,k) = \sum_{\substack{n_1+n_2=n-1 \\ \ell_1 \cdot \ell_2 + k_1 + k_2 = k}} F(n_1,k_1,\ell_1) \cdot G(n_2,k_2,\ell_2)$$

$$F(n,k,\ell) = \begin{cases} A(n,k) & \text{if } n = \ell \\ 0 & \text{otherwise} \end{cases}$$

$$G(n,k,\ell) = \begin{cases} A(n,k-n) & \text{if } n = \ell \\ 0 & \text{otherwise} \end{cases}$$

This will be part of current PhD student Emile Nadeau's thesis. Watch this space! Hopefully, "mesh" statistics as well.

ARTIFICIAL INTELLIGENCE?

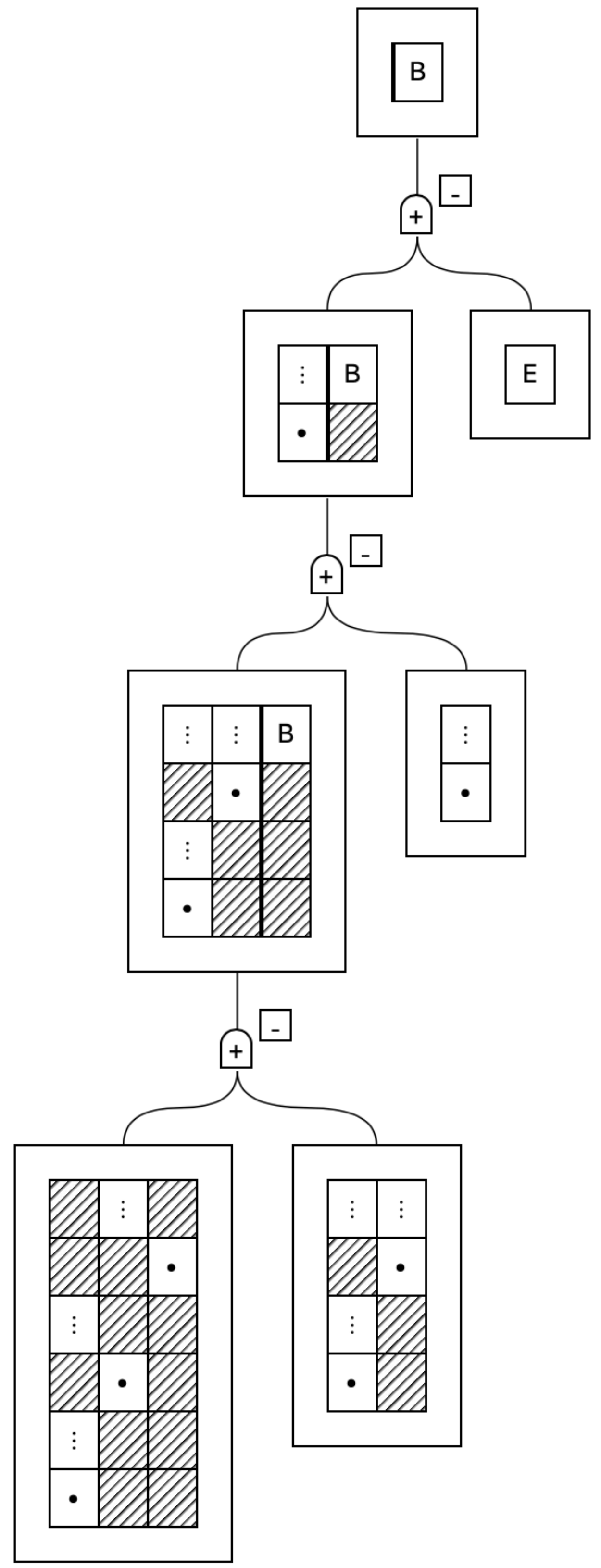
Not in the current open implementation. There is an MSc student, Ragnar Ardal, graduating in January 2020, who has implemented *proof-number-search* for choosing “good tilings” in the universe to work on. It has shown great promise for many classes, sometimes reducing The search time by a factor of 10. It does also get stuck searching down paths that will never lead to a success. In those cases the current naive brute-force breadth first search is faster.

This is the first tool we try to add from AI

OTHER DOMAINS: SET PARTITIONS

Set Partitions avoiding 1223,1231,1233,1234

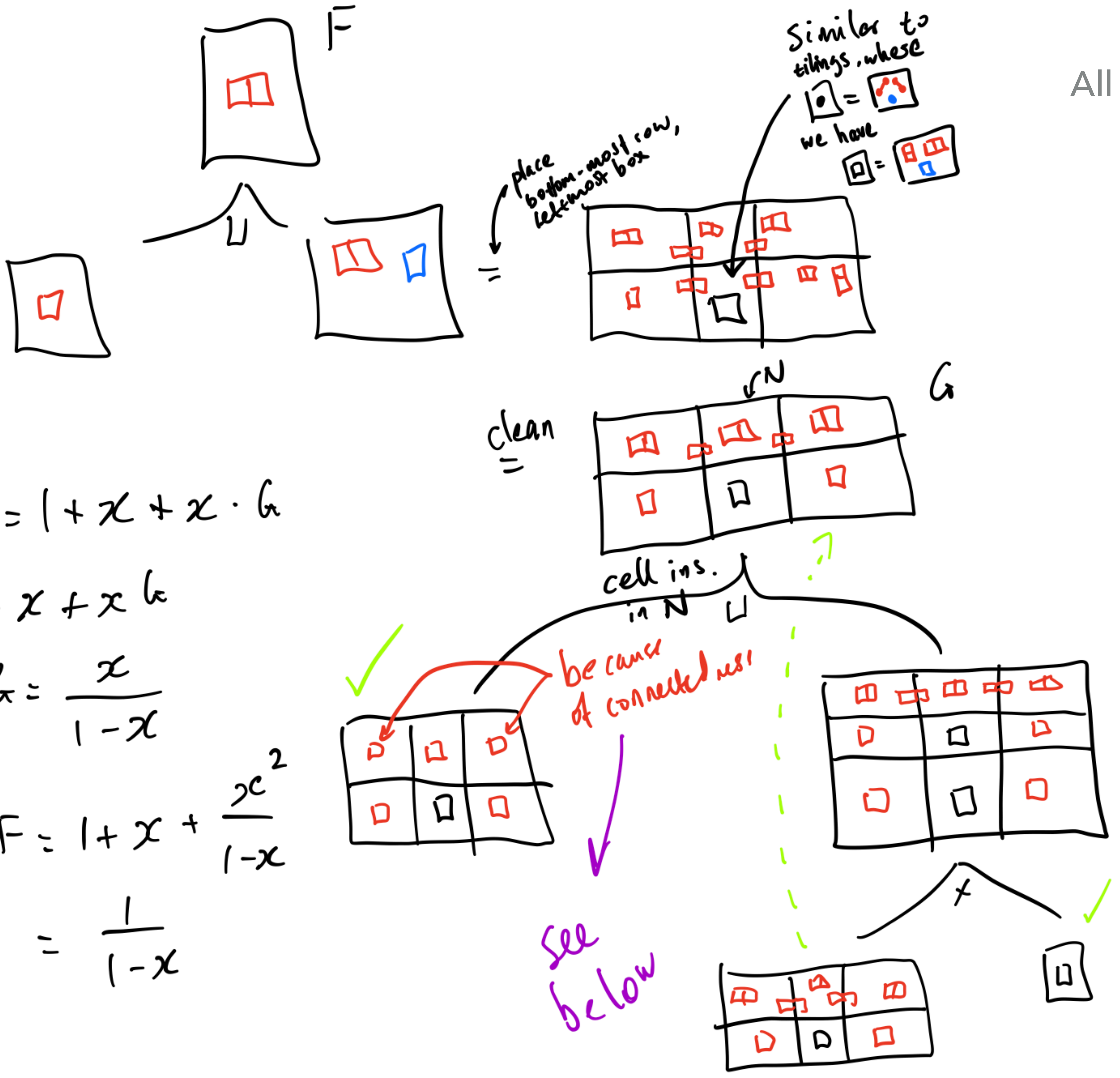
Proof tree for set partitions avoiding 1223,1231,1233,1234



We advised an undergraduate project which looked at applying our framework to pattern avoiding set partitions.

You can see many results on ComboPal.

OTHER DOMAINS: POLYOMINOES



$$F = 1 + x + x \cdot G$$

$$G = x + xG$$

$$\Rightarrow G = \frac{x}{1-x}$$

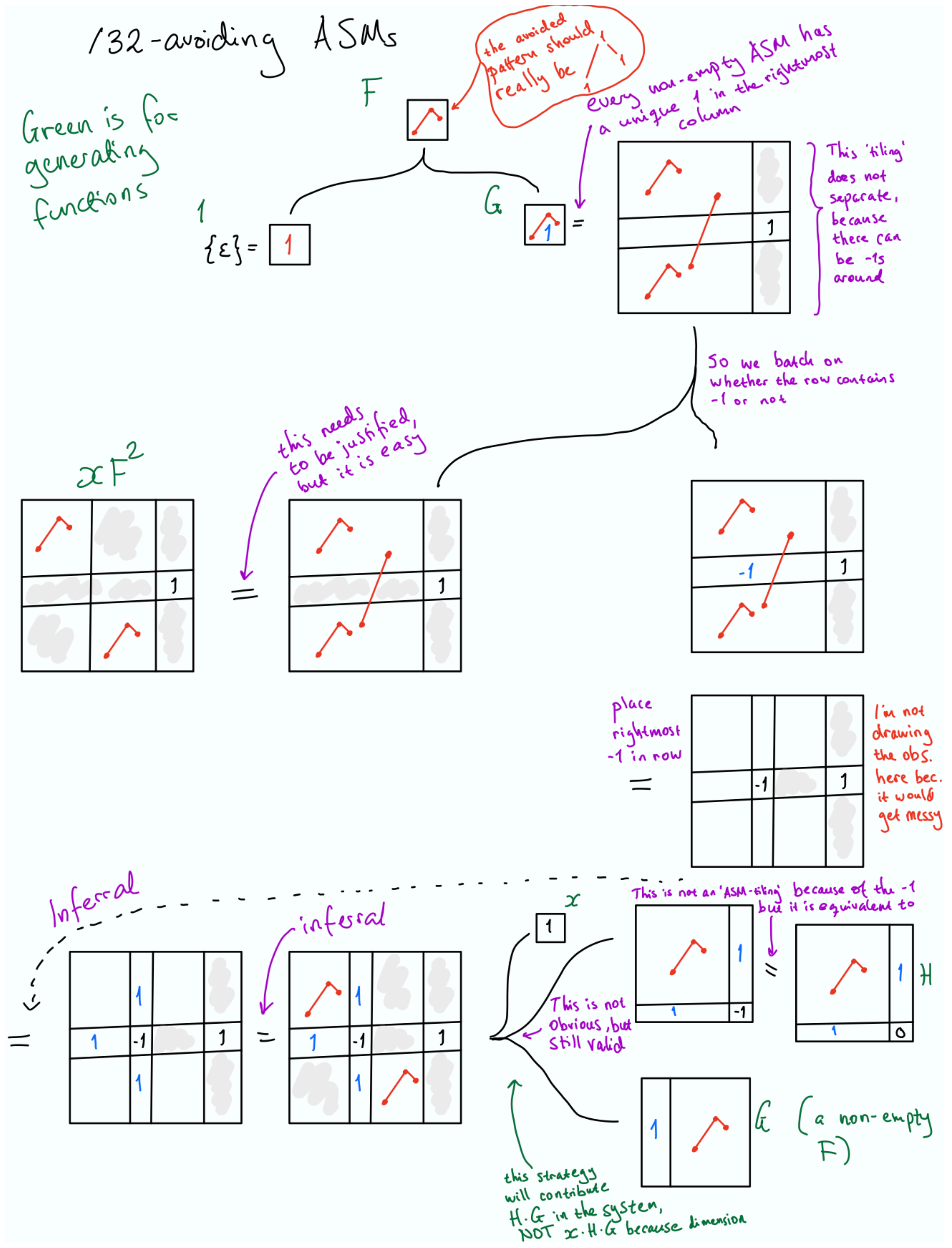
$$\Rightarrow F = 1 + x + \frac{x^2}{1-x}$$

$$= \frac{1}{1-x}$$

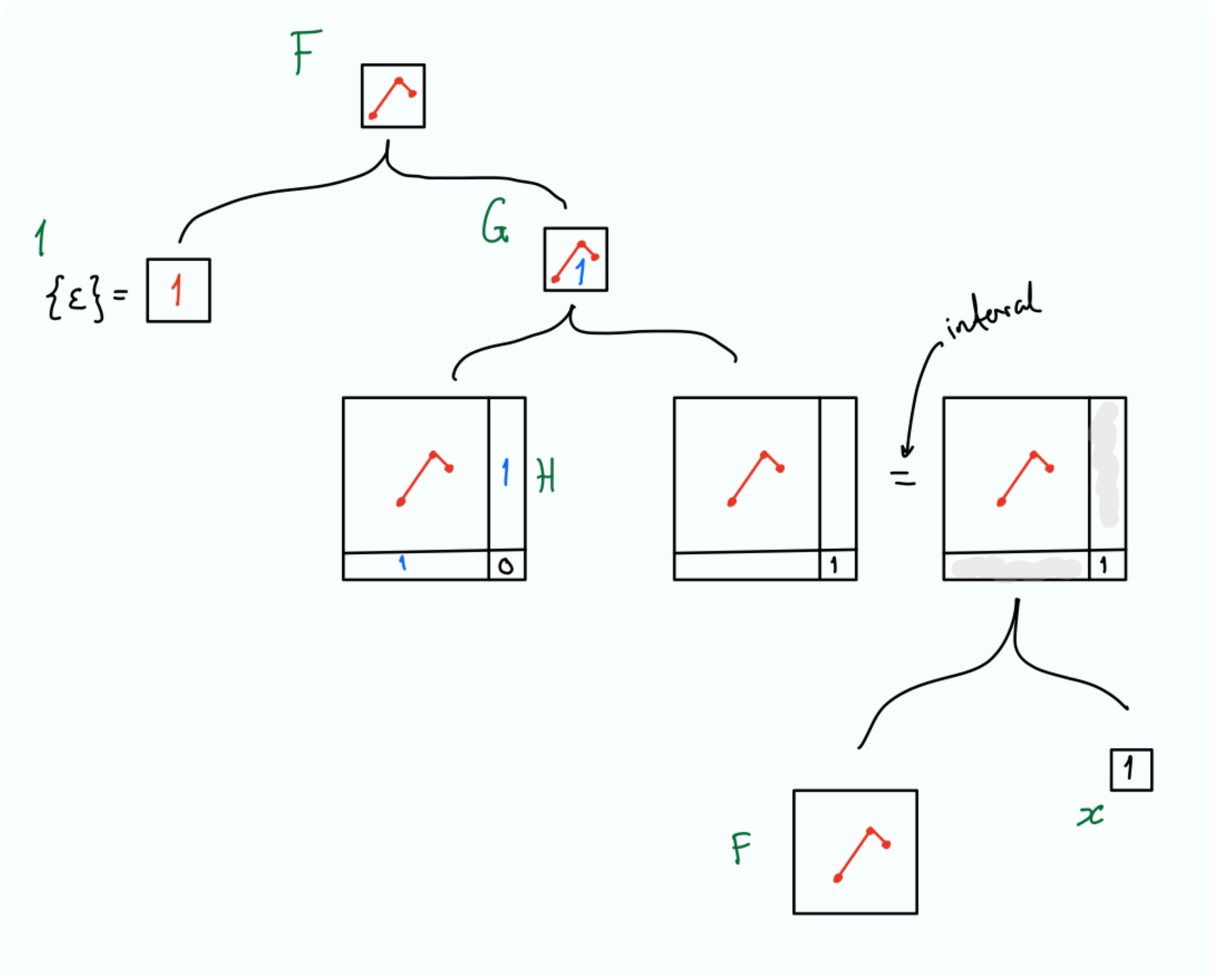
All polyominoes: ???

OTHER DOMAINS: ALTERNATING SIGN MATRICES (ASM)

All ASMs: $$\prod_{k=0}^{n-1} \frac{(3k+1)!}{(n+k)!}$$



132-avoiding ASMs
Green is for generating functions
the avoided pattern should really be
F
G
1
{ε} =
every non-empty ASM has a unique 1 in the rightmost column
This 'tiling' does not separate, because there can be -1s around
So we branch on whether the row contains -1 or not
this needs to be justified, but it is easy
xF²
place rightmost -1 in row
I'm not drawing the obs. here bec. it would get messy
Inferral
inferral
x
This is not an 'ASM-tiling' because of the -1 but it is equivalent to
H
This is not obvious, but still valid
G (a non-empty F)
this strategy will contribute H·G in the system, NOT x·H·G because dimension



Combinatorial ~~specifications~~ forests …

OTHER DOMAINS AND THE FUTURE

What are other good domains? Necessary properties

- Many (infinite) problems of a similar type
- Some way of representing the state in code
- Strategies that can be coded

Future

- Can find bijections if two inputs have isomorphic proof trees
- Turning the output into human-readable text
- Random sampling of objects
- Proving formally (in Coq, LEAN, …) the underlying framework, and that the output of a strategy is correct. This would formally verify any proof tree we produce