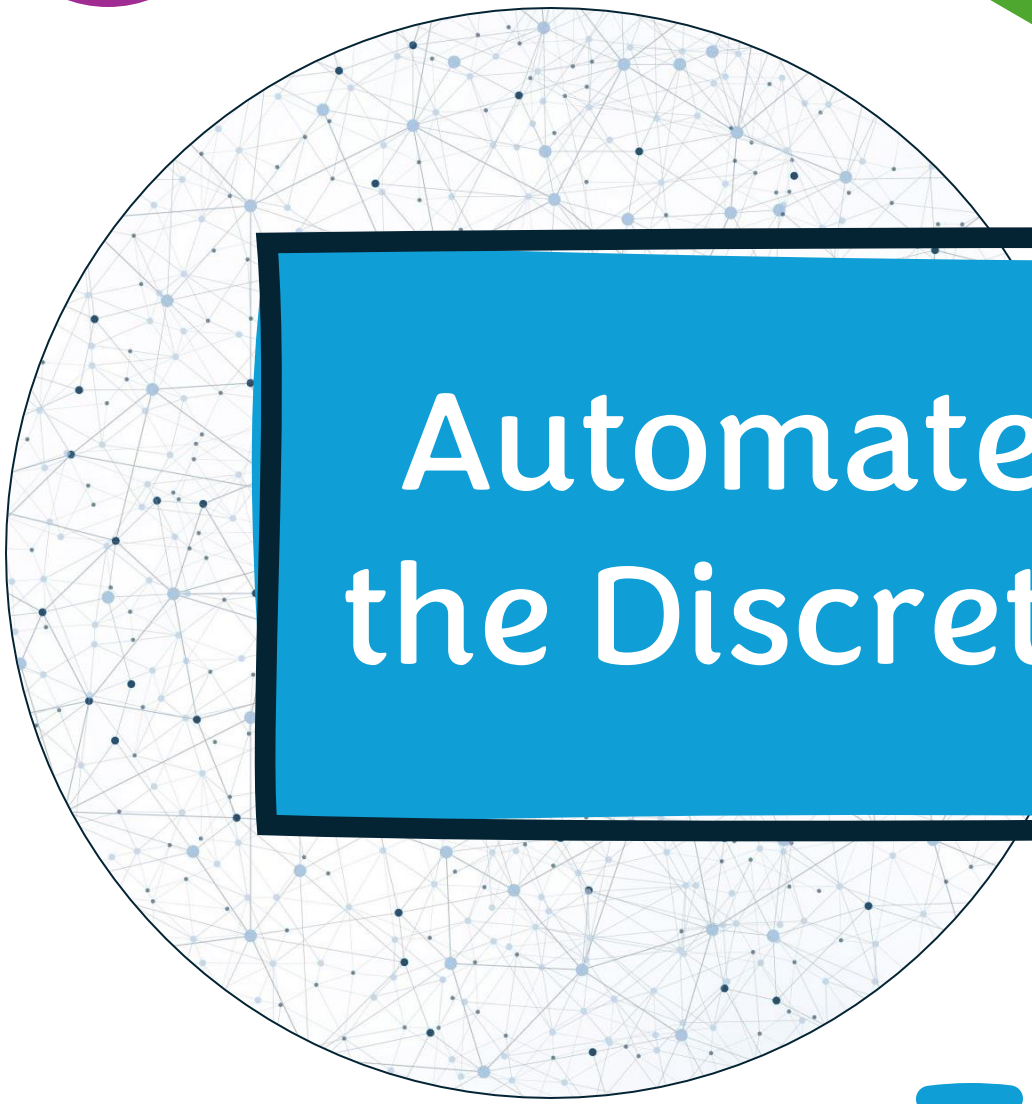


Automated Reasoning for the Discrete Mathematician

Bernardo Subercaseaux

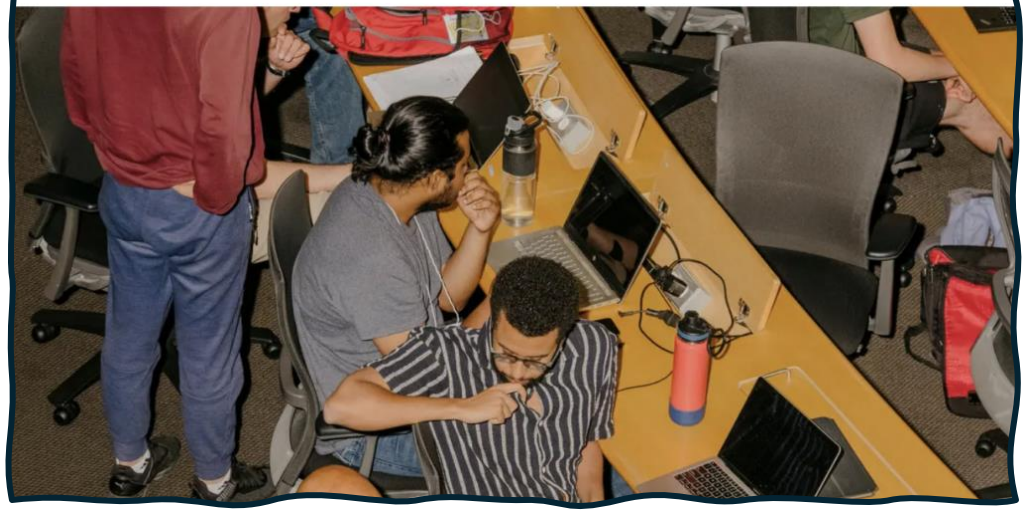
Carnegie Mellon University

The elephant in the room…

A.I. Is Coming for Mathematics, Too

For thousands of years, mathematicians have adapted to the latest advances in logic and reasoning. Are they ready for artificial intelligence?

Give this article



NYTimes

AMS American Mathematical Society
201 Charles Street | Providence, RI 02904-2213 USA

Advisory Group on Artificial Intelligence and the Mathematical Community
www.ams.org/artificial-intelligence

Questions Artificial Intelligence Raises for the Mathematics Profession

AMS

JUNE 8, 2024 | 12 MIN READ

AI Will Become Mathematicians' 'Co-Pilot'

Fields Medalist Terence Tao explains how proof checkers and AI programs are dramatically changing mathematics

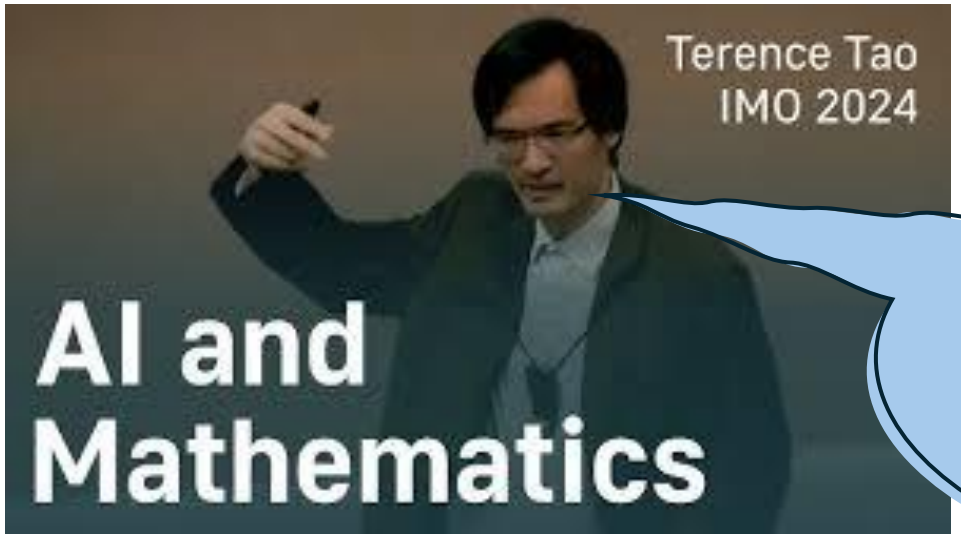
BY CHRISTOPH DRÖSSER EDITED BY GARY STIX

Scientific American

"What's most exciting to me is modeling **new modes of human-machine collaboration**.

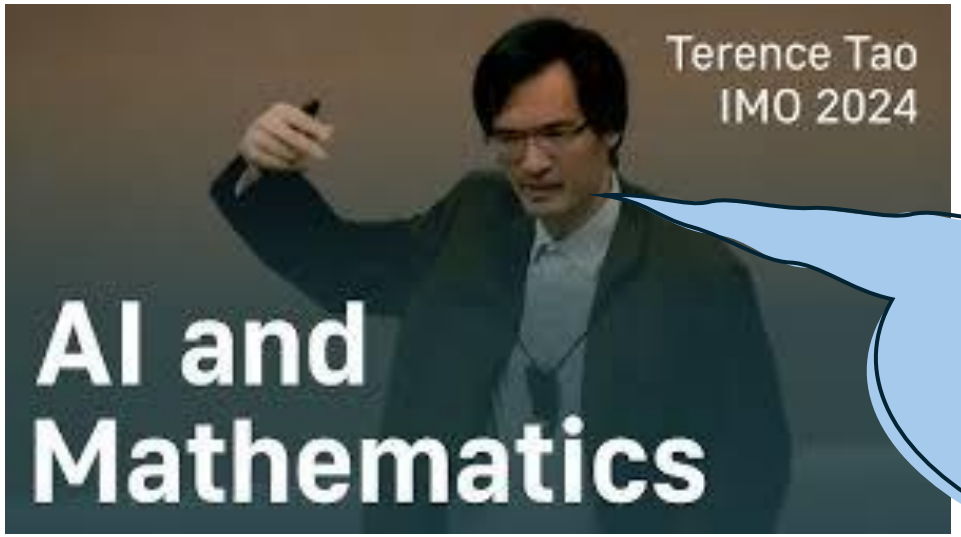
I don't look to use these as a replacement for human mathematicians, but as **a force multiplier**."

- Jordan Ellenberg



Computation has been assisting math since the beginning !

- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)
- Computer Algebra Systems (Maple, SAGE, Mathematica)
- SAT solvers
- Formal theorem provers (Lean)
- LLMs?



Computation has been assisting math since the beginning !

- Abacus
- Tables (Logarithm, Primes, OEIS, Elliptic curves, Groups)
- Computer Algebra Systems (Maple, SAGE, Mathematica)
- **SAT solvers**
- Formal theorem provers (Lean)
- LLMs?

SAT solvers have been impactful in math!

- **(2014)** Boolean Erdős Discrepancy Problem
- **(2016)** Boolean Pythagorean Triples
- **(2018)** Schur Number 5
- **(2019)** Keller's Conjecture
- **(2023) Packing Chromatic Number of $\mathbb{Z}^2$**
- **(2024)** Empty Hexagon every 30 Points

These all follow a common pattern:

Using SAT solvers to tackle a hard combinatorial problem that brute force computation (i.e., backtracking) would take forever on

npactful in math!

oblem

- **(2023)** Packing Chromatic Number of Z^2
- **(2024)** Empty Hexagon every 30 Points

npactful in math!

oblem

- **(2023)** Packing Chromatic
- **(2024)** Empty Hexagon ev

These all follow a common pattern:

Using SAT solvers to tackle a hard combinatorial problem that brute force computation (i.e., backtracking) would take forever on

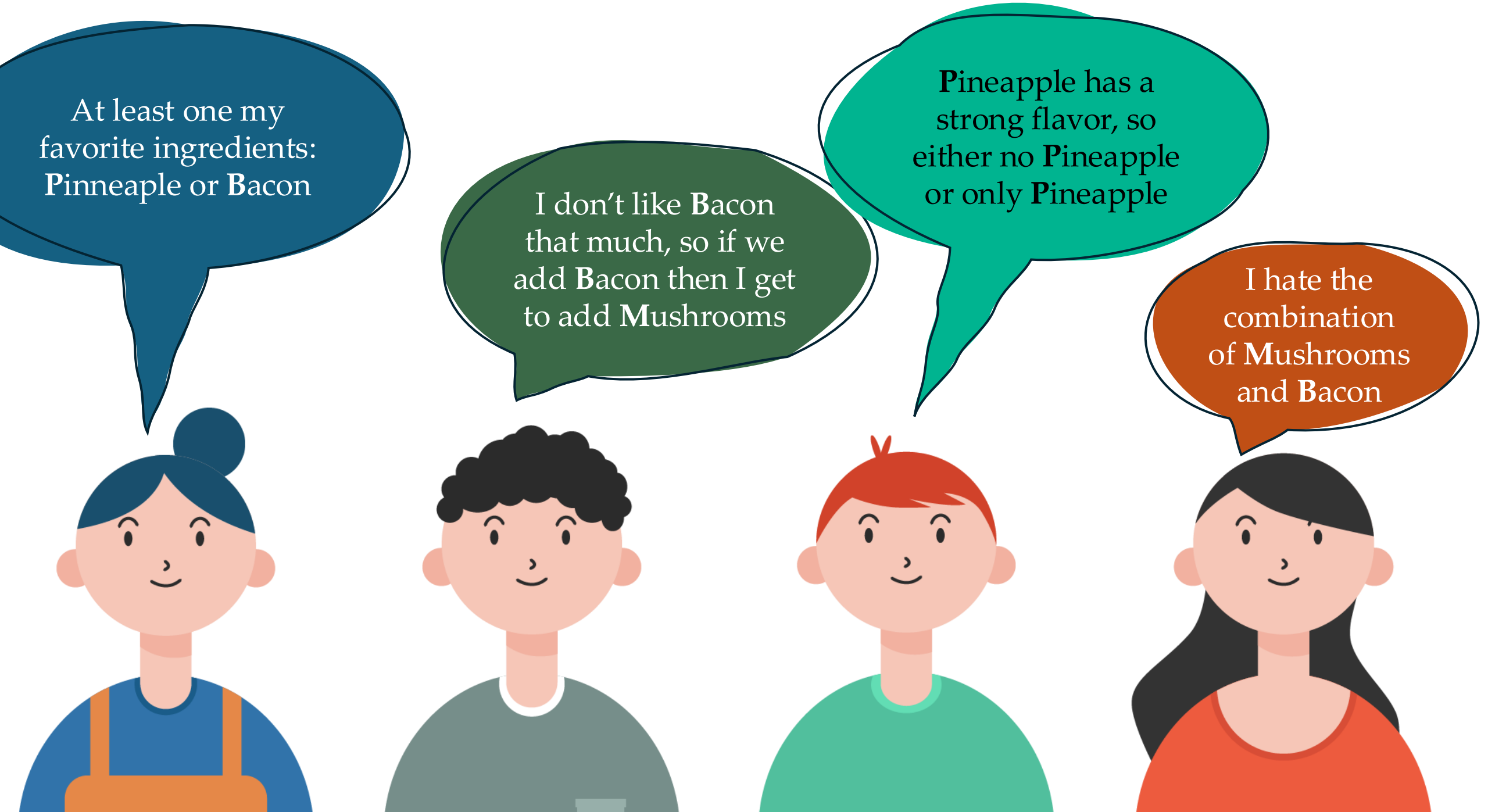
What this talk is about:

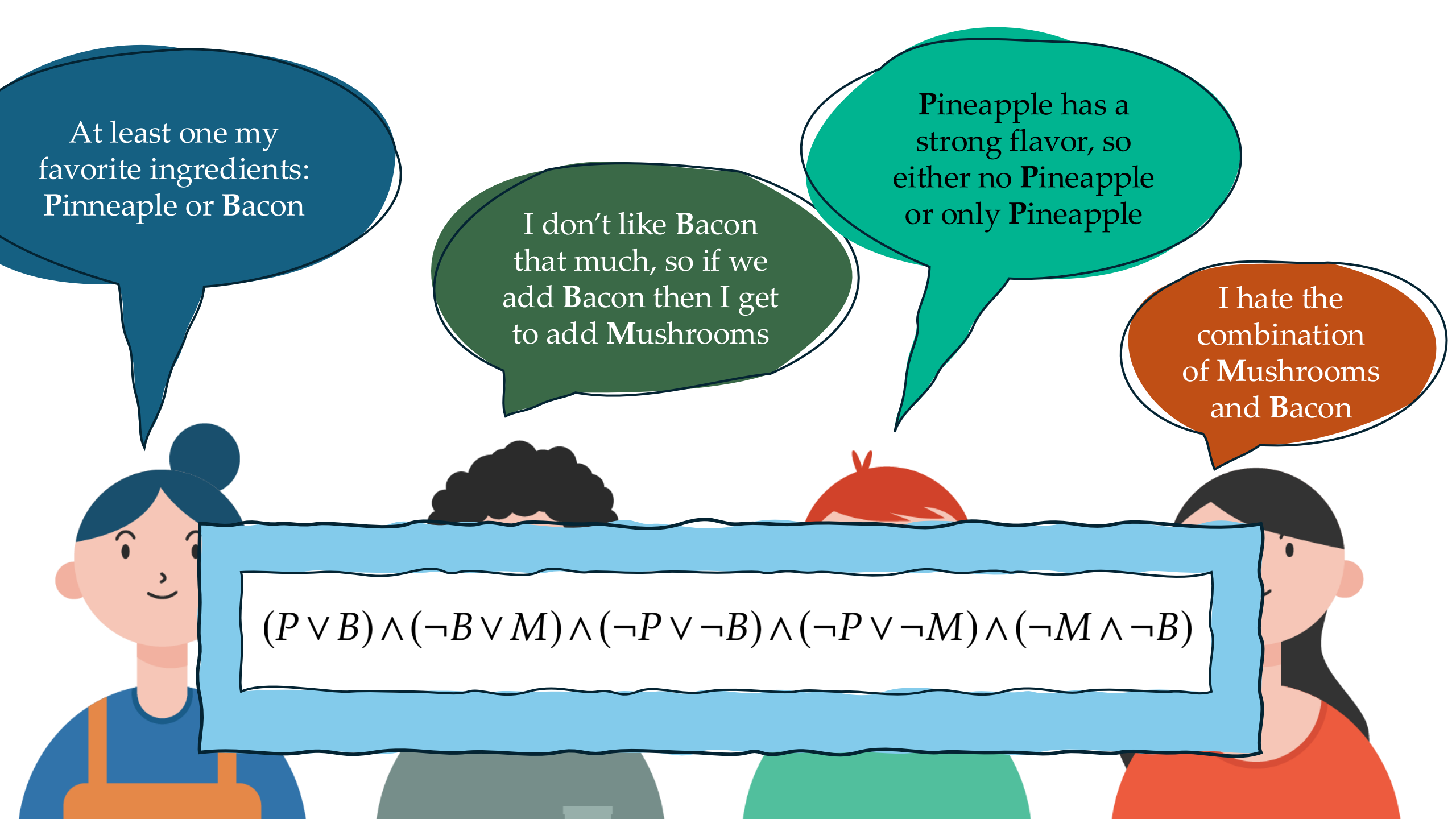
SAT solvers can be used at different stages of mathematical research; to build examples, get ideas and elicit conjectures.

In the next 20 minutes

I will try to convince you that SAT solvers are:

- A powerful tool for assisting discrete math research
- Easy to use
- Very cool, and can benefit from your math skills
- Free of hallucinations ;)


At least one my favorite ingredients: Pinneaple or Bacon
I don't like Bacon that much, so if we add Bacon then I get to add Mushrooms
Pineapple has a strong flavor, so either no Pineapple or only Pineapple
I hate the combination of Mushrooms and Bacon


At least one my favorite ingredients: Pinneaple or Bacon
I don't like Bacon that much, so if we add Bacon then I get to add Mushrooms
Pineapple has a strong flavor, so either no Pineapple or only Pineapple
I hate the combination of Mushrooms and Bacon
$(P \vee B) \wedge (\neg B \vee M) \wedge (\neg P \vee \neg B) \wedge (\neg P \vee \neg M) \wedge (\neg M \wedge \neg B)$

Quick Q&A

Q: What is a SAT solver?
A: A highly optimized program for deciding if a Boolean formula is satisfiable

Q: A Boolean formula?

A: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$

Q: Satisfiable?

A: Yes! $(x_1, x_2, x_3) := (0, 1, 0)$

Q: And how do I write this on my computer ?

A:

test.cnf

```
p cnf 3 2
1 -2 3 0
-1 2 0
```

Quick Q&A

Q: And then what?

A:

or

Q: Wait so I have to manually write the formula in a file?!
A: No! The standard way is to write code, in any language of your choice, that generates the formula.

Quick Q&A

Q: Wait but isn't SAT an NP-complete problem?!

A: Exactly!

It's hard !

Writing your own efficient algorithm is a large engineering challenge

It's universal !

Can represent a lot of different math problems

E.g., Graphs, matroids, groups, geometry

A toy example: **Sudoku**

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Unsolved Sudoku

4	2	*6*	5	*7*	1	*3*	9	*8*
8	*5*	*7*	2	*9*	3	*1*	*4*	6
1	3	*9*	*4*	6	*8*	*2*	7	*5*
9	*7*	1	*3*	*8*	*5*	6	*2*	*4*
5	4	*3*	*7*	*2*	*6*	*8*	1	9
6	*8*	2	*1*	*4*	*9*	7	*5*	*3*
7	9	*4*	*6*	3	*2*	*5*	8	*1*
2	*6*	*5*	8	*1*	4	*9*	*3*	7
3	1	*8*	9	*5*	7	*4*	6	*2*

Solved Sudoku

Each cell must contain a number from {1, …, 9}

Each number from {1, …, 9} needs to appear exactly once per:
- Row
- Column
- 3x3 *major subgrid*

A toy example: **Sudoku**

Variables: $x_{i,j,n}$

"cell *(i, j)* gets number *n*"

sudoku_sat.py

```python
def encode(clues):
    """
    Encode a sudoku puzzle as a SAT problem.
    clues : 9x9 int matrix, non-zero values represent clues
    """
    enc = modeler.Modeler() # my library

    # create variables
    for i in range(9):
        for j in range(9):
            for n in range(1, 10):
                enc.add_var(f"x_{i, j, n}")
```

Sudoku Constraints

Each cell must contain a number from {1, …, 9}

$$\forall i \in [9], \forall j \in [9], \left(\bigvee_{n \in [9]} x_{i,j,n} \right)$$

Respect clues!

$$(x_{1,4,5}) \wedge (x_{2,1,8}) \wedge \cdots$$

sudoku_sat.py

```python
# at least one number per cell
for i in range(9):
    for j in range(9):
        enc.add_clause([f"x_{i, j, n}" for n in range(1, 10)])

# respect clues
for i in range(9):
    for j in range(9):
        if clues[i][j] != 0:
            enc.add_clause([f"x_{i, j, clues[i][j]}"])
```

Sudoku Constraints

How to encode "**Exactly one** of the variables $x_1, x_2, \ldots, x_n$ is true"?

At least one $\left(\bigvee_{i=1}^{n} x_i \right)$

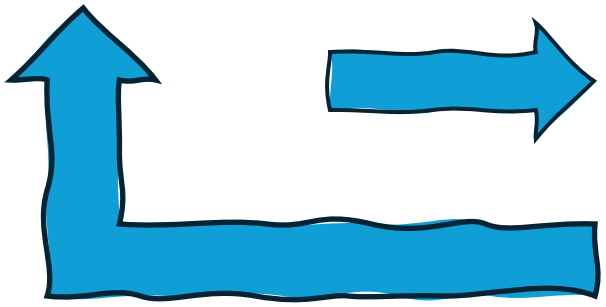
At most one (naïve) $\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j)$

SAT encodings can be interesting ;)

If all we have are the variables $x_1, x_2, \ldots, x_n$
then AMO($x_1, x_2, \ldots, x_n$) requires $O(n^2)$ clauses

Introduce "auxiliary" variable y that *summarizes* information

Suppose y summarizes whether a variable in $\{x_1, x_2, x_3\}$ is true



$$\text{AMO}(y, x_4, \ldots, x_n)$$

$$\text{AMO}(x_1, x_2, x_3, \neg y)$$

SAT encodings can be interesting ;)

$$\mathrm{AMO}(x_1, \ldots, x_n) \cong$$
$$\mathrm{AMO}(x_1, x_2, x_3, \neg y) \wedge \mathrm{AMO}(y, x_4, \ldots, x_n)$$

Number of clauses $f(n)$ satisfies $f(n) = f(4) + f(n\text{-}2)$

so we have $f(n) = O(n)$

Cardinality constraints included in libraries

sudoku_sat.py

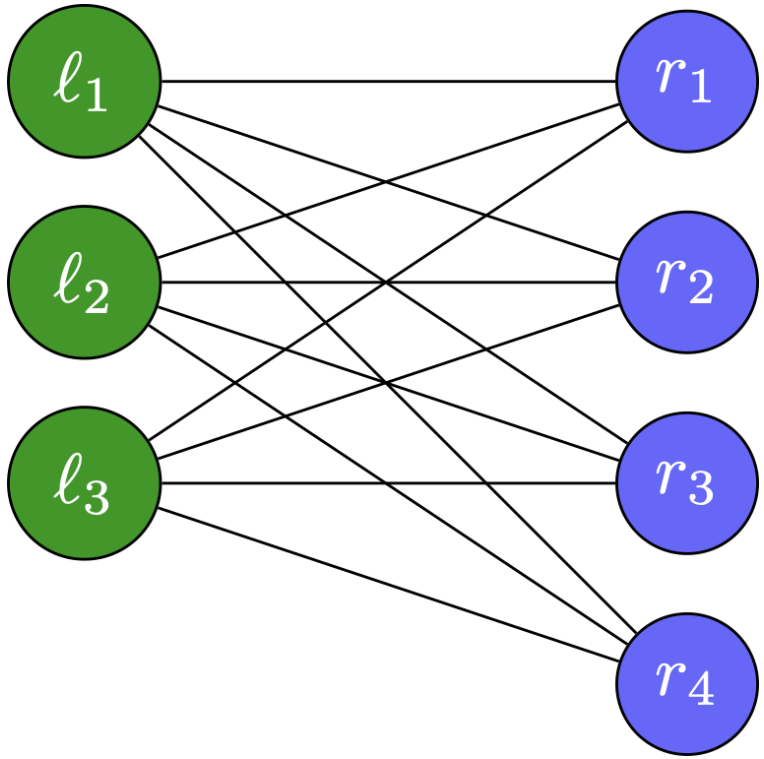
```python
# exactly-one constraints
for n in range(1, 10):
    # rows
    for i in range(9):
        enc.exactly_one([f"x_{i, j, n}" for j in range(9)])

    # cols
    for j in range(9):
        enc.exactly_one([f"x_{i, j, n}" for i in range(9)])

    # sub_grids
    sub_grids = [[[] for sj in range(3)] for si in range(3)]
    for i in range(9):
        for j in range(9):
            sub_grids[i//3][j//3].append((i, j))
    for si in range(3):
        for sj in range(3):
            enc.exactly_one([f"x_{cell, n}" for cell in sub_grids[si][sj]])
```

Okay, now on to some math examples!

Main idea: to gain insight on a problem, we will search for finite objects.
The (non-)existence of such object will be encoded in a Boolean formula

Mantel's problem

What's the maximum number of edges on a triangle-free graph on n vertices?



Could we have more edges?

Is there a non-isomorphic graph that also achieves this?

Mantel's problem

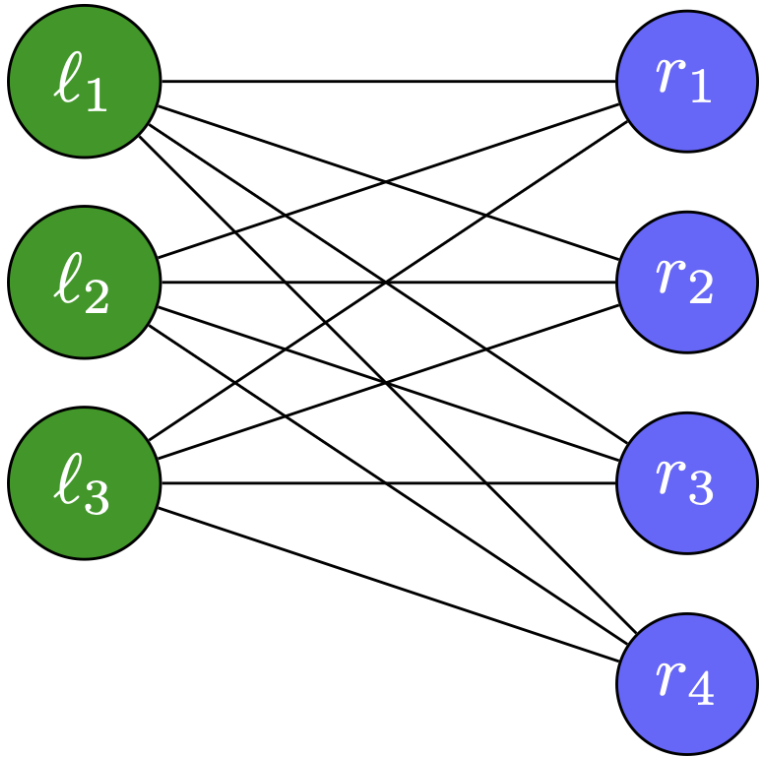
What's the maximum number of edges on a triangle-free graph on n vertices?

mantel_sat.py

```python
def encode_mantel(n, m):
    """ Encodes whether a graph with n vertices and m edges can have no triangles
    """
    enc = modeler.Modeler()
    for u in range(n):
        for v in range(u+1, n):
          enc.add_var(f"e_{u, v}") # edge variables

    # no triangles
    for u in range(n):
        for v in range(u+1, n):
            for w in range(v+1, n):
                enc.add_clause([f"-e_{u, v}", f"-e_{v, w}", f"-e_{u, w}"])

    edge_vars = [f"e_{u, v}" for u in range(n) for v in range(u+1, n)]
    enc.at_least_k(edge_vars, m)
```

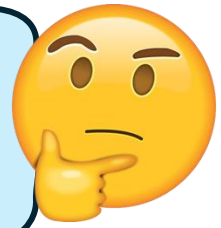
Mantel's problem

What's the maximum number of edges on a triangle-free graph on **n** vertices?



Could we have more edges?

Is there a non-isomorphic graph that also achieves this?



Mantel's problem & Symmetry Breaking

If sum of degrees is $2m$, some vertex has degree at least $2m/n$

WLOG, that's vertex 0, and that it's connected to 1, 2, …, ceil($2m/n$)

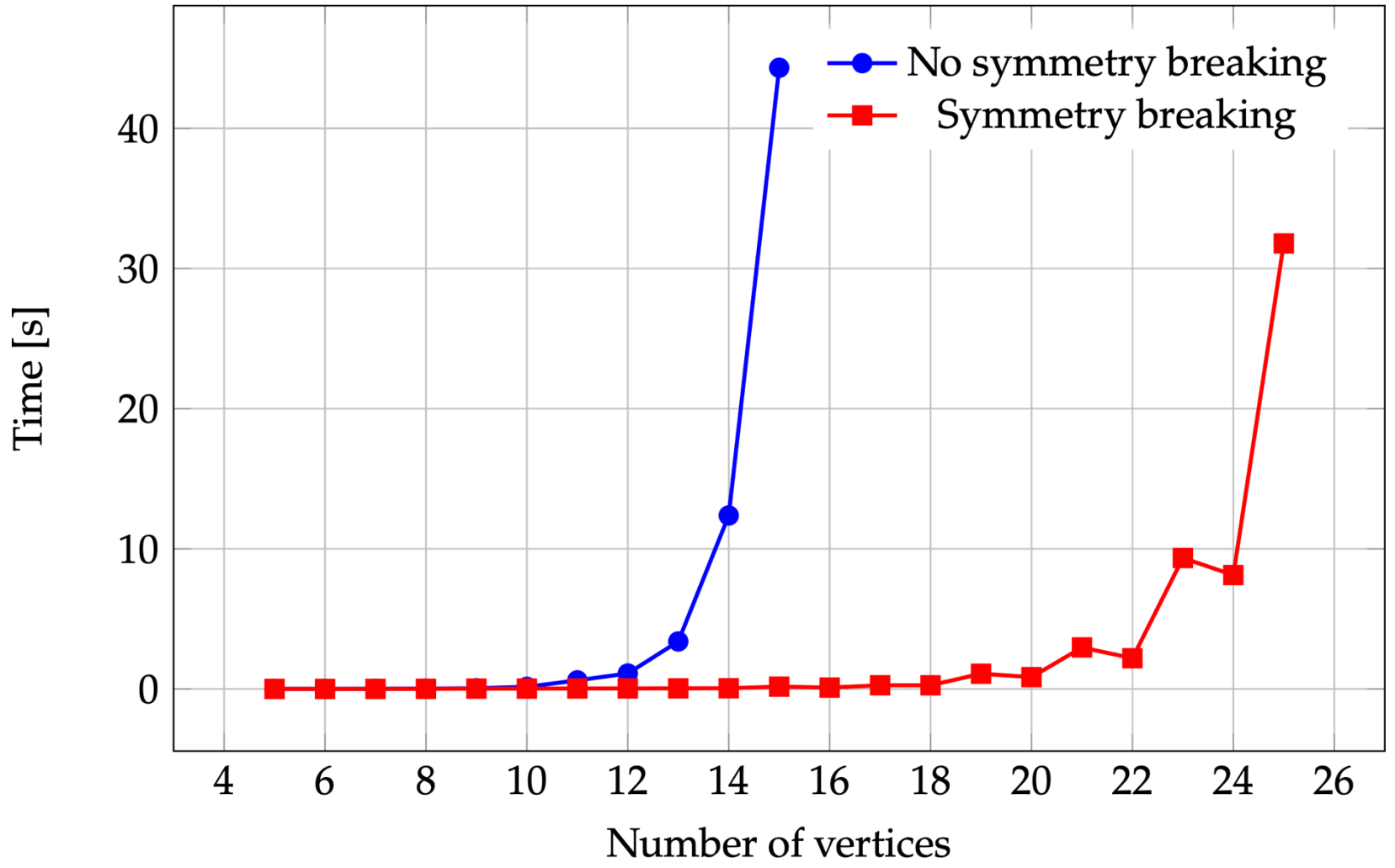
mantel_sat.py

```python
def encode_mantel(n, m):
    [...]
    # symmetry breaking
    for i in range(1, 2*math.ceil(m/n)+1):
        enc.add_clause([f"e_{0, i}"])
```

This is not enough to break all symmetries of the problem, but if we do a more complex encoding that breaks all symmetries

Then solution is unique!

Mantel's problem & Symmetry Breaking



Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

lagrange_sat.py

```python
# "multiplication" table
for i in elements:
    for j in elements:
        for k in elements:
            enc.add_var(f"m_{i, j, k}", description=f"{i} * {j} = {k}")

# i * j has a unique answer
for i in elements:
    for j in elements:
        enc.exactly_one([f"m_{i, j, k}" for k in elements])

# Neutral element is assumed to be 0 wlog. 0 * i = i * 0 = i
for i in elements:
    enc.add_clause([f"m_{0, i, i}"])
    enc.add_clause([f"m_{i, 0, i}"])
```

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

lagrange_sat.py

```python
# Inverse elements:
for i in elements:
    for j in elements:
        enc.add_var(f"inv_{i, j}", description=f"{j} is the inverse of {i}")
for i in elements:
    enc.add_clause([f"inv_{i, j}" for j in elements])
    for j in elements:
        # inv_{i, j} -> m_{i, j, 0} & m_{j, i, 0}
        enc.add_clause([f"-inv_{i, j}", f"m_{i, j, 0}"])
        enc.add_clause([f"-inv_{i, j}", f"m_{j, i, 0}"])

# associativity: (i * j) * k = i * (j * k)
for i in elements:
    for j in elements:
        for k in elements:
            for l in elements:
                for m in elements:
                    for s in elements:
                        enc.add_clause([f"-m_{i, j, l}", f"-m_{j, k, m}", f"-m_{l, k, s}", f"m_{i, m, s}"])
```

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

lagrange_sat.py

```
# desired subgroup:
for i in elements:
    enc.add_var(f"e_{i}", description=f"element {i} is in the subgroup")

# neutral element is in the subgroup
enc.add_clause([f"e_{0}"])

# inverses are in the subgroup
for i in elements:
    for j in elements:
        # e_i & inv_{i, j} -> e_j
        enc.add_clause([f"-e_{i}", f"-inv_{i, j}", f"e_{j}"])

# closure
for i in elements:
    for j in elements:
        for k in elements:
            enc.add_clause([f"-e_{i}", f"-e_{j}",  f"-m_{i, j, k}", f"e_{k}"])

# subgroup size
enc.exactly_k([enc.v(f"e_{i}") for i in elements], S)
```

Lagrange's theorem

For every subgroup **H** of a finite group **G**, |**H**| divides |**G**|

What about the "converse"?

Is there a group **G** on n elements, and some divisor d of n such that **G** has **no** subgroup of size d?

Yes! A group on 12 elements without subgroups of size 6

Discrete Geometry

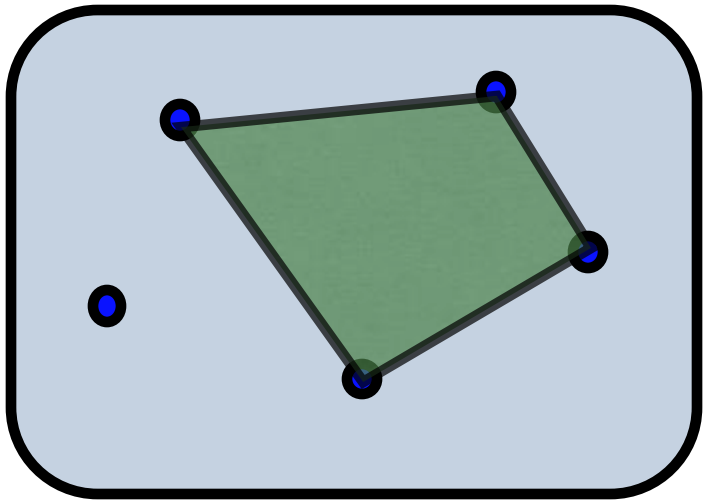
How many points, without 3 on a line, guarantee a **convex quadrilateral**?

Answer: 5, Klein, *"Happy Ending Theorem"*

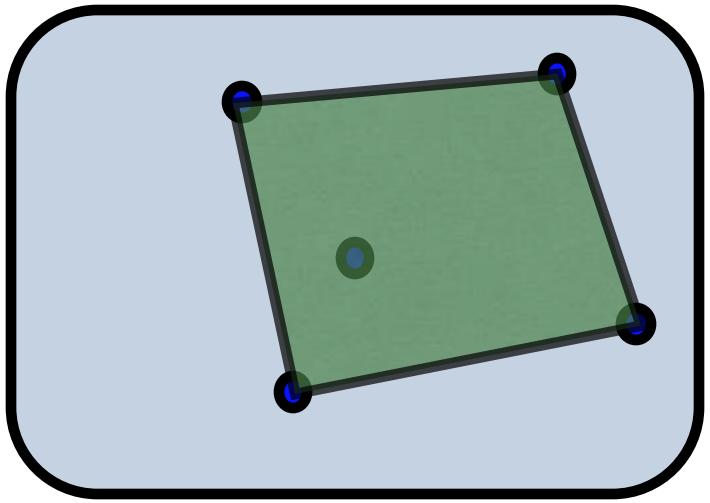
Discrete Geometry

How many points, without 3 on a line, guarantee a **convex quadrilateral**?

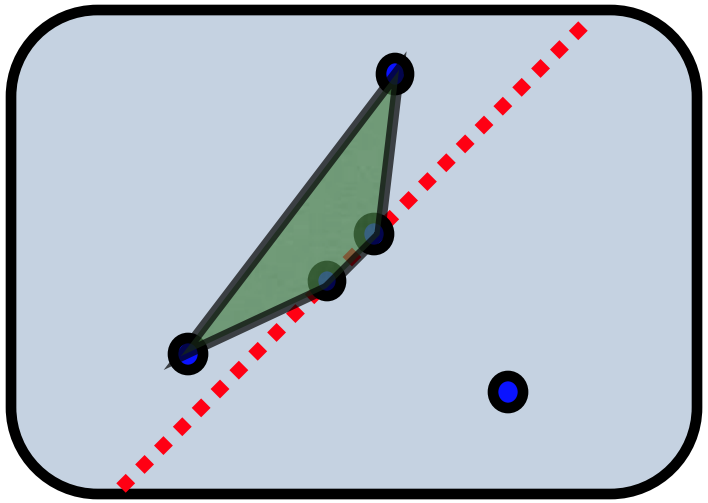
Answer: 5, Klein, *"Happy Ending Theorem"*



Case 1: 5 points in Convex Hull



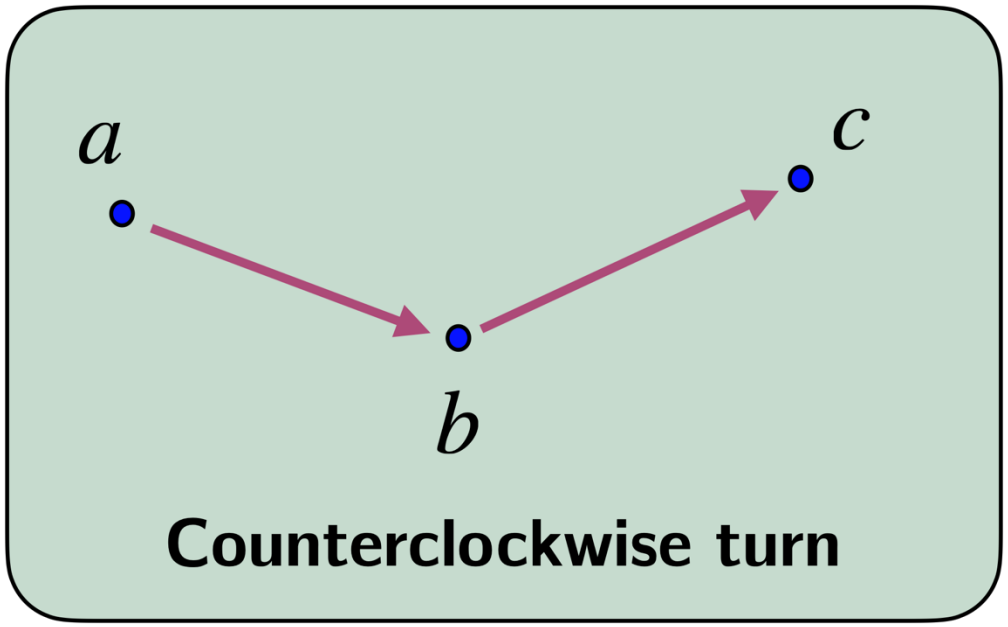
Case 2: 4 points in Convex Hull



Case 3: 3 points in Convex Hull

Discrete Geometry

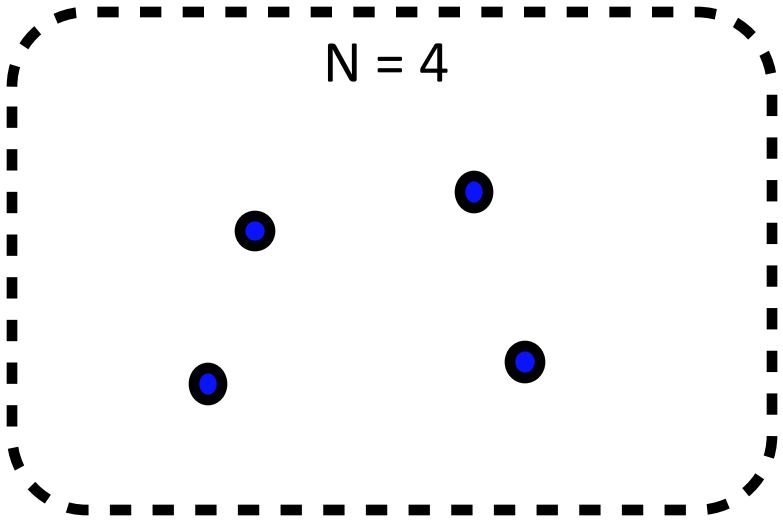
We can define variables $s_{(a,\,b,\,c)}$ that represent whether a,b,c is a CC-turn



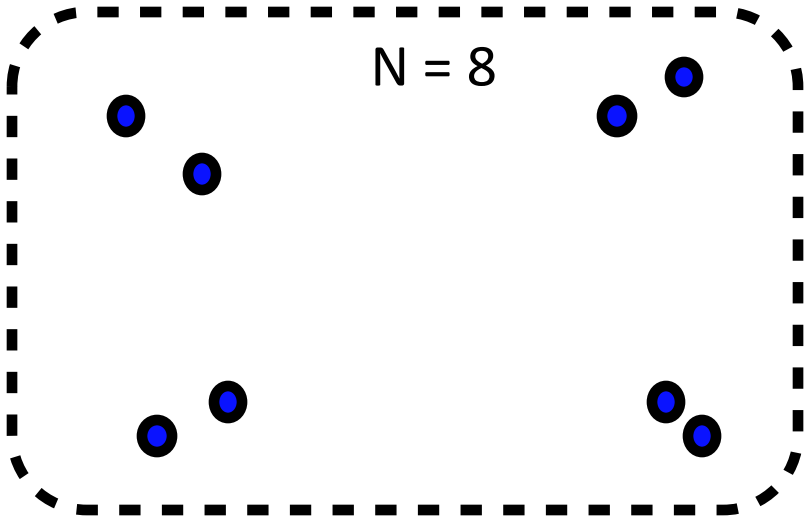
$$\det \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{bmatrix} > 0$$

Problem: Minimizing Convex Pentagons

Given an integer N, what is, $\mu_5(N)$ the minimum number of convex pentagons we can get by placing N points in the Euclidean plane without 3 on a line?



Answer = 0



Answer = 0

N	$\mu_5(N)$
4	0
8	0
9	1
15	**77**

Discrete Geometry: My research

How to minimize # of convex pentagons among N points in general position?

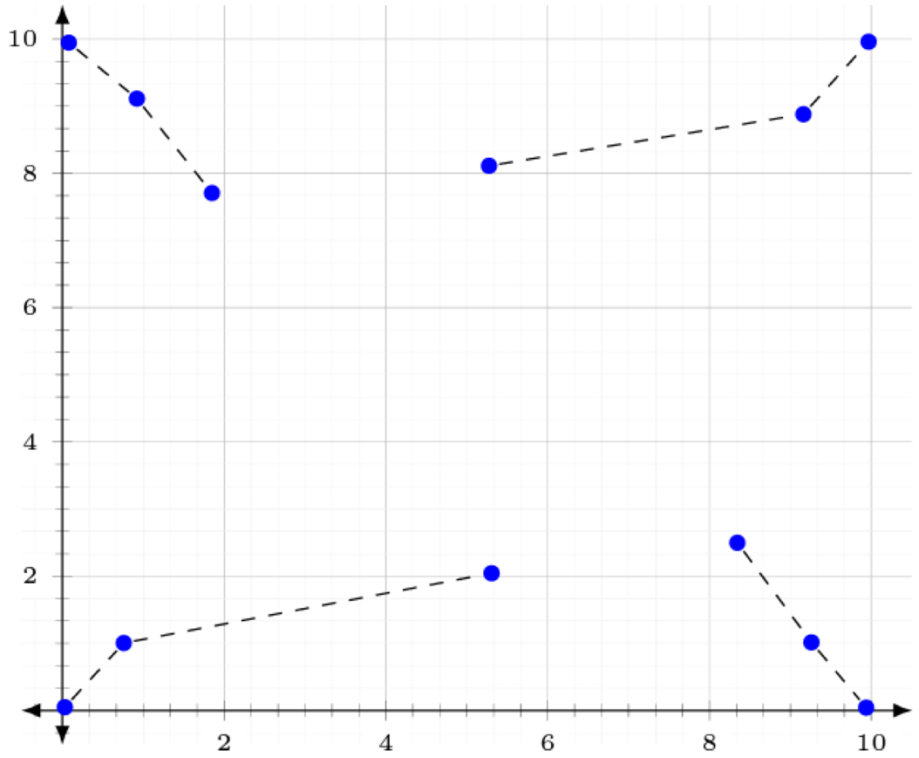
N	Best	Time
9	1	
10	2	
11	7	
12	12	
13	27	0.01 s
14	42	0.01 s
15	77	0.01 s
16	112	0.02 s

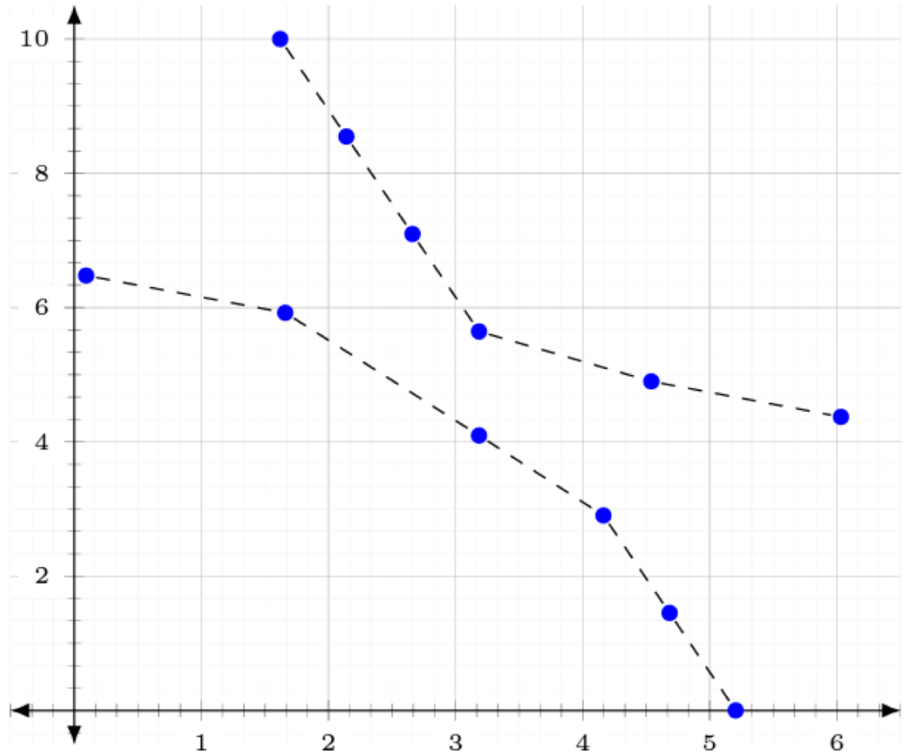
$$\mu_5(N) = \binom{\lfloor N/2 \rfloor}{5} + \binom{\lceil N/2 \rceil}{5}$$

N	Best	Time
		12 s
		472 s
		64 s
		5269 s
27	3289	1556 s
28	4004	1792 s
29	5005	467 s
30	6007	18244 s

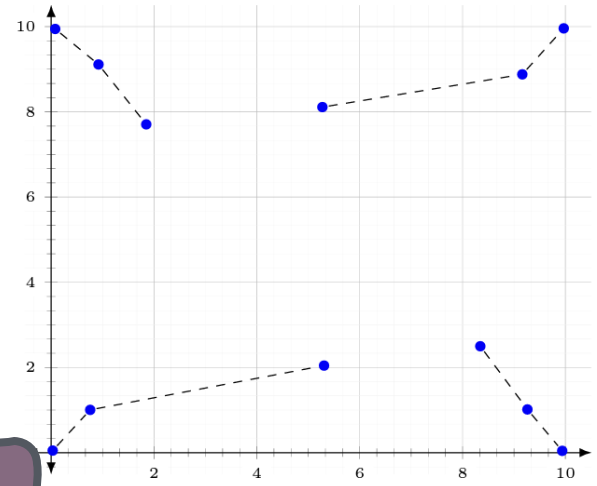
Discrete Geometry: My research

How to minimize # of convex pentagons among N points in general position?

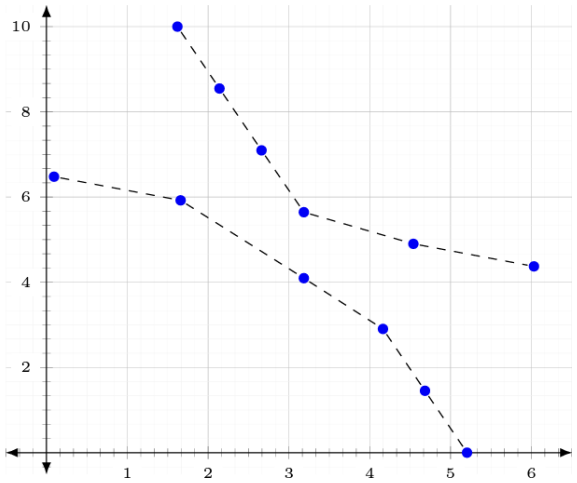




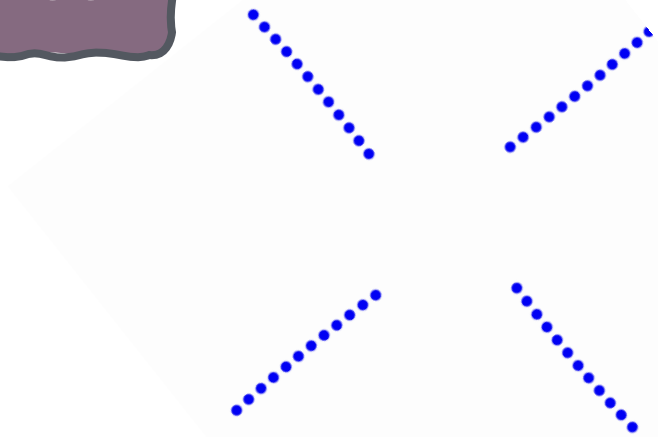
Generalize them as constructions!



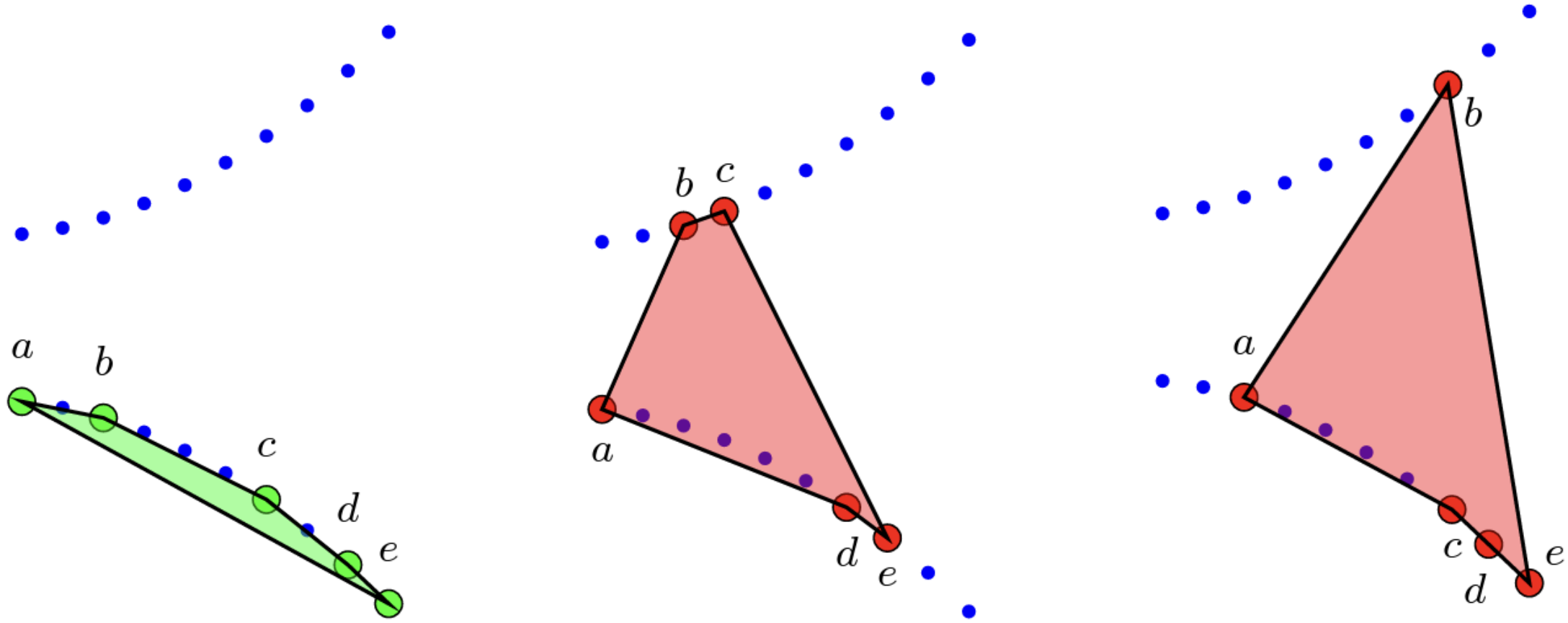
Pinwheel



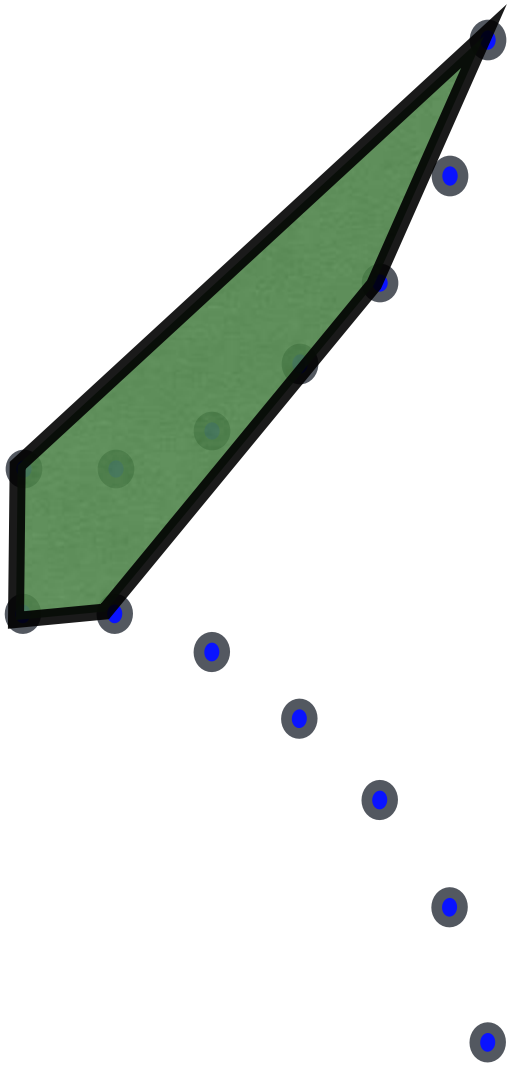
Parabolic

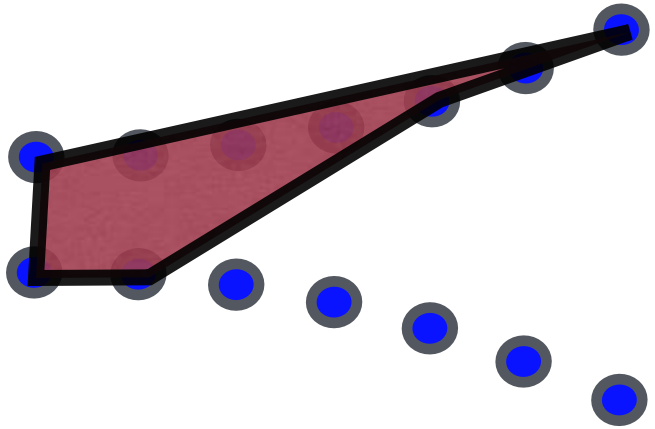


Parabolic Construction



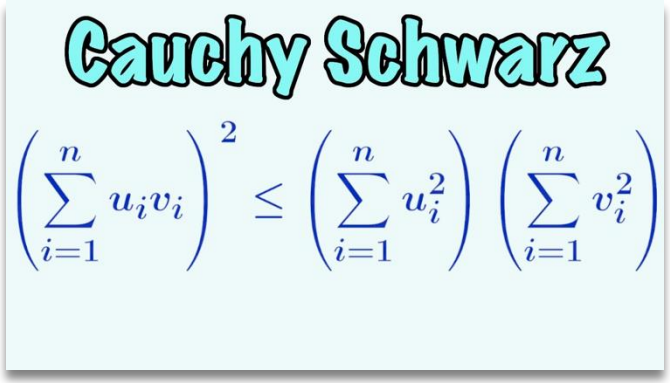
We need to be careful though!

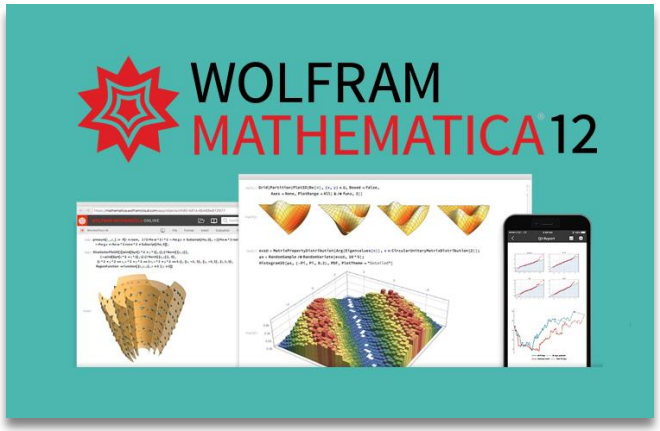


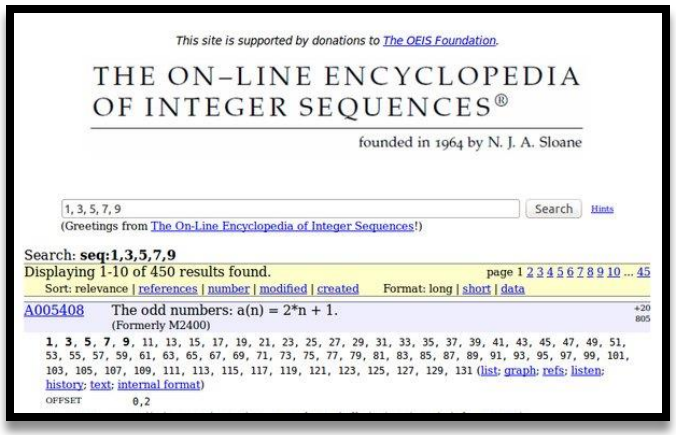


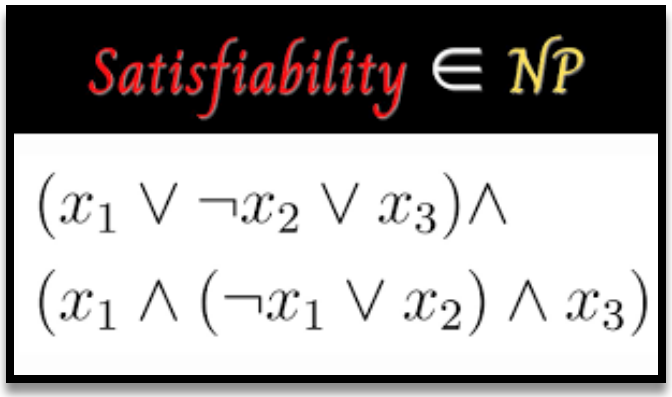
$$p_i^{\top} = \left(i, 2 + \frac{i^2}{n^2}\right), \forall i \in \left[\left\lfloor \frac{n}{2} \right\rfloor\right] \quad \text{and} \quad p_i^{\perp} = \left(i, -2 - \frac{i^2}{n^2}\right), \forall i \in \left[\left\lceil \frac{n}{2} \right\rceil\right]$$

Conclusion: A mathematician's toolbox

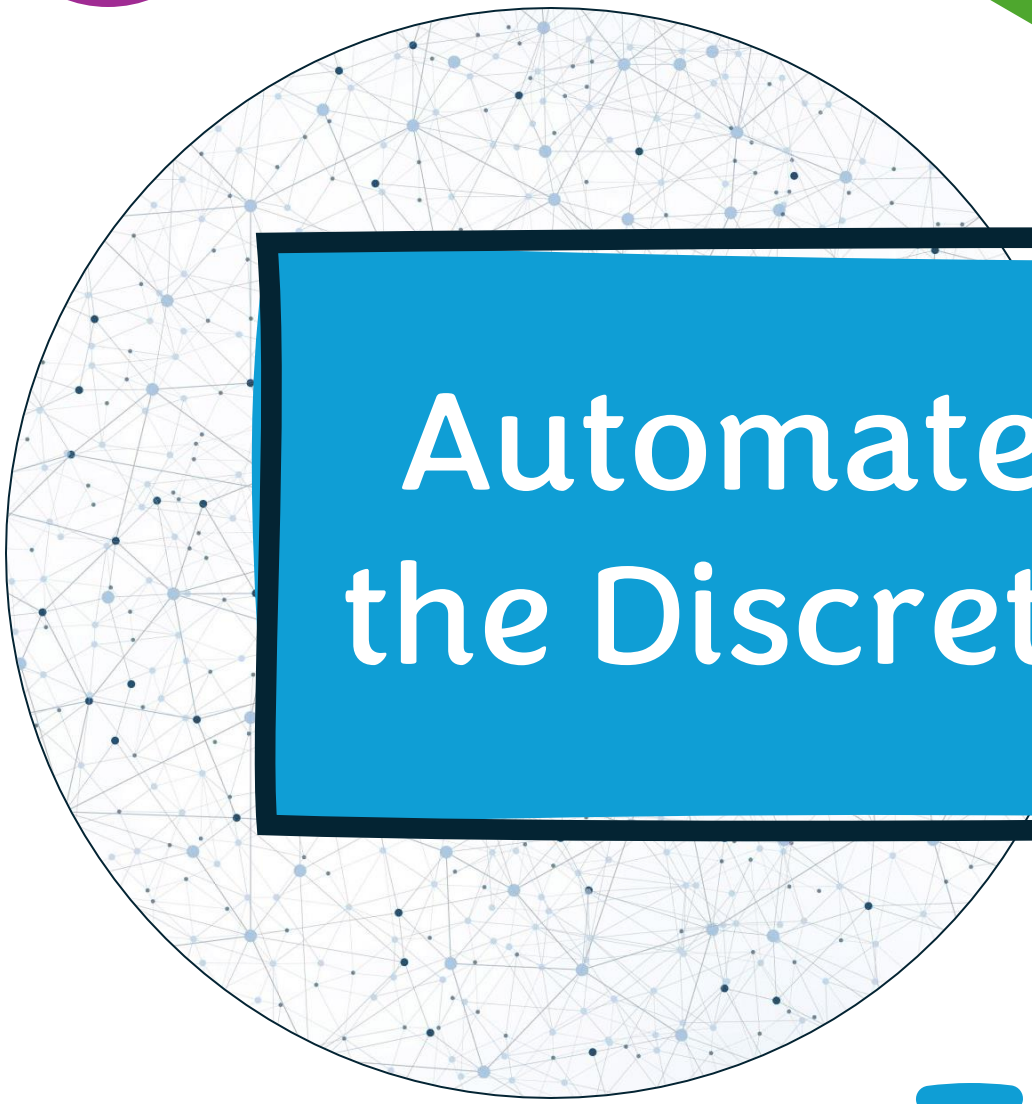








SAT Solvers?



Automated Reasoning for the Discrete Mathematician

Bernardo Subercaseaux

Carnegie Mellon University