# Complexity, Undecidability and Tilings

Chaim Goodman-Strauss
Univ Arkansas
strauss@uark.edu

# Why are tiling puzzles difficult?

Why are tiling puzzles difficult?

(And how difficult are they anyway?)

There is nothing particularly special about tilings in this regard:

There is nothing particularly special about tilings in this regard:

*Every* combinatorial system, if it is rich enough, will pose computationally complex problems, in a manner we can define precisely.

There is nothing particularly special about tilings in this regard:

*Every* combinatorial system, if it is rich enough, will pose computationally complex problems, in a manner we can define precisely.

In essence, the question is:
**Can our system emulate arbitrary computations?**

There is nothing particularly special about tilings in this regard:

*Every* combinatorial system, if it is rich enough, will pose computationally complex problems, in a manner we can define precisely.

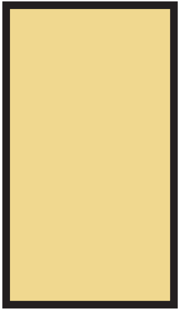
In essence, the question is:
**Can our system emulate arbitrary computations?**

Tilings give a nice model, and in turn, the theory of computation illuminates some classical tiling problems.

Consider a simple question:

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

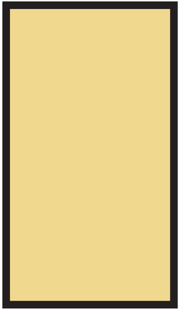


trivial example

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

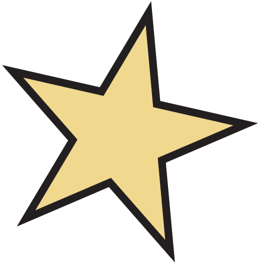


trivial example

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

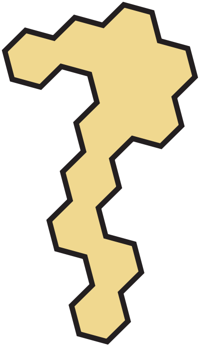


certainly not

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

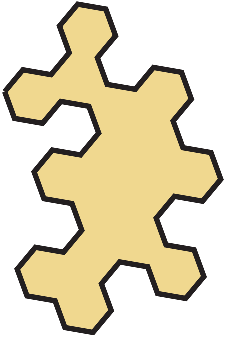


Myers (2003)

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

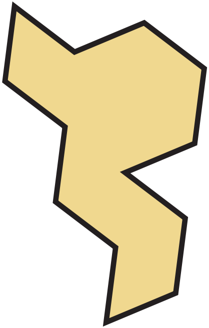


Myers (2003)

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

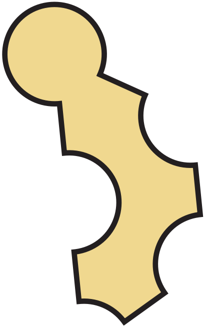


Myers (2003)

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

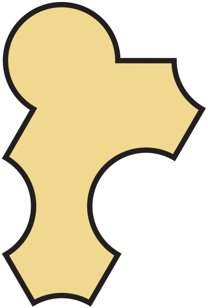


Mann (2007)

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

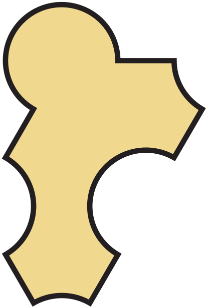


Mann (2007)

Consider a simple question:

Can this tile be used to form a tiling of the entire plane?

How can you tell?

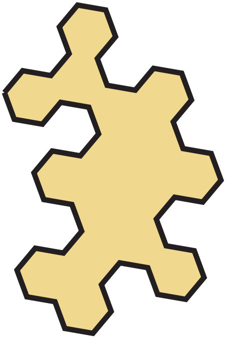


Mann (2007)

Is there a general method to tell whether or not a given tile admits a tiling?

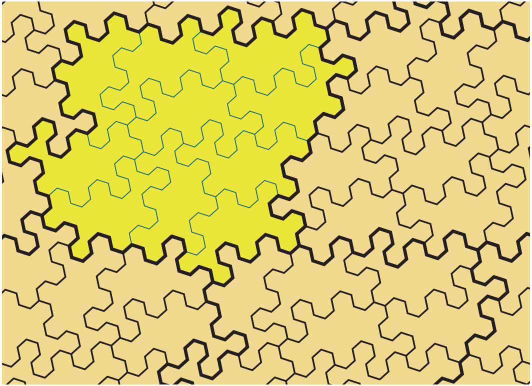
Many of these examples are very badly behaved.

This example (Myers 2003) has *isohedral number* 10, the current world record



Many of these examples are very badly behaved.

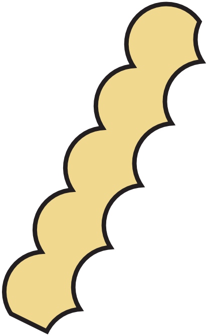
That is, the tile can form periodic tilings, but the tiles fall into at least ten orbits; equivalently, the smallest possible fundamental domain has ten tiles!



An isohedral number 10 example

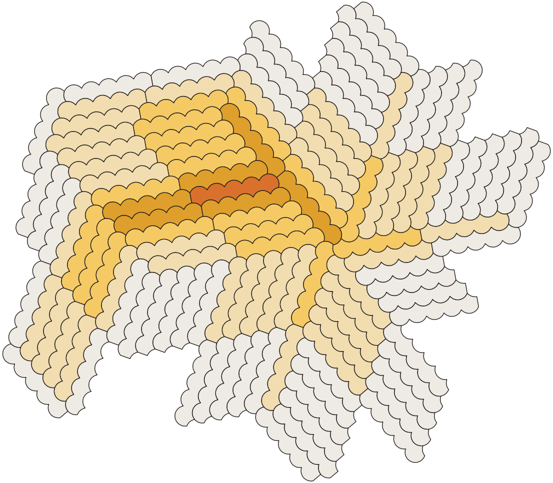
Many of these examples are very badly behaved.

This example (Mann 1999) doesn't admit a tiling, but you can form quite large patches before things fall apart.



Many of these examples are very badly behaved.

It has *Heesch number* 5, the current world record– it can form a patch with five “coronas" and no more.



Many of these examples are very badly behaved.

There is no reason to suppose these are the worst possible examples—

Many of these examples are very badly behaved.

There is no reason to suppose these are the worst possible examples—

We have no idea (really) how they work, or what obstructions, if any, there are to creating increasingly terrible tiles.

Many of these examples are very badly behaved.

There is no reason to suppose these are the worst possible examples—

We have no idea (really) how they work, or what obstructions, if any, there are to creating increasingly terrible tiles.

Again, *Is there a way to tell whether a given tile admits a tiling of the entire plane?*

The obvious algorithm to try is

*See how far you can get!*

The obvious algorithm to try is

*See how far you can get!*

More precisely, we might enumerate all possible configurations by the tile, trying to cover larger and larger regions.

The obvious algorithm to try is
*See how far you can get!*

More precisely, we might enumerate all possible configurations by the tile, trying to cover larger and larger regions.

• If the tile does *not* admit a tiling, then there will be some largest region that we can cover.

**Thm:** If we can cover arbitrarily large regions, we can cover the entire plane.

The obvious algorithm to try is

*See how far you can get!*

More precisely, we might enumerate all possible configurations by the tile, trying to cover larger and larger regions.

• If the tile does *not* admit a tiling, then there will be some largest region that we can cover.

But this algorithm will never halt if the tile *does* admit a tiling.

The obvious algorithm to try is
*See how far you can get!*

More precisely, we might enumerate all possible configurations by the tile, trying to cover larger and larger regions.

• If the tile does *not* admit a tiling, then there will be some largest region that we can cover.

But this algorithm will never halt if the tile *does* admit a tiling.
So we make a modification.

The obvious algorithm to try is
*See how far you can get!*

More precisely, we might enumerate all possible configurations by the tile, trying to cover larger and larger regions.

• If the tile does *not* admit a tiling, then there will be some largest region that we can cover.

• As we proceed, we check to see if any of our configurations could be a fundamental domain. If the tile admits a *periodic* tiling, we can discover this too.

The obvious algorithm to try is

*See how far you can get!*

More precisely, we might enumerate all possible configurations by the tile, trying to cover larger and larger regions.

• If the tile does *not* admit a tiling, then there will be some largest region that we can cover.

• As we proceed, we check to see if any of our configurations could be a fundamental domain. If the tile admits a *periodic* tiling, we can discover this too.

This works fine— the algorithm will halt with a yes or no answer, so long as nothing falls through the gaps— so long as every tile either doesn't admit a tiling or admits a periodic tiling.

Surely this is the case! ?

Surely this is the case! ?

Surely it is impossible for there to exist an *aperiodic* tile, that is, a tile that does admit a tiling of the plane, but never a periodic tiling.

Surely this is the case! ?

Surely it is impossible for there to exist an *aperiodic* tile, that is, a tile that does admit a tiling of the plane, but never a periodic tiling.

Such a tile would have to force some sort of bad behavior at all scales.

Surely this is the case! ?

Surely it is impossible for there to exist an *aperiodic* tile, that is, a tile that does admit a tiling of the plane, but never a periodic tiling.

Such a tile would have to force some sort of bad behavior at all scales.

And surely there is a general procedure (or theorem, or theory, or algorithm) that can determine whether a given monotile admits a tiling of the plane.

Surely this is the case! ?

Surely it is impossible for there to exist an *aperiodic* tile, that is, a tile that does admit a tiling of the plane, but never a periodic tiling.

Such a tile would have to force some sort of bad behavior at all scales.

And surely there is a general procedure (or theorem, or theory, or algorithm) that can determine whether a given monotile admits a tiling of the plane.

How hard can this be?

Surely this is the case! ?

Surely it is impossible for there to exist an *aperiodic* tile, that is, a tile that does admit a tiling of the plane, but never a periodic tiling.

Such a tile would have to force some sort of bad behavior at all scales.

And surely there is a general procedure (or theorem, or theory, or algorithm) that can determine whether a given monotile admits a tiling of the plane.

How hard can this be?

The examples we see here should make us cautious.

The algorithm we outlined (enumerate all configurations, covering larger and larger disks, until we either crash, or find a fundamental domain) succeeds if there is no aperiodic tile.

If there is no aperiodic tile, then this algorithm solves both the Domino Problem and the Period Problem for any given monotile:

*The Domino Problem for Monotiles:*

Given a tile, does it fail to admit a tiling?

*The Period Problem for Monotiles:*

Given a tile, does it admit a periodic tiling?

That is:

The Period Problem is decidable for monotiles

There is no aperiodic monotile

The Domino Problem is decidable for monotiles

That is:

The Period Problem is undecidable for monotiles $\Searrow$

There is an aperiodic monotile

The Domino Problem is undecidable for monotiles $\Nearrow$

Moreover, suppose there is a bound $H$ on Heesch number; i.e., suppose every tile that does not admit a tiling has Heesch number less than $H$.

Moreover, suppose there is a bound $H$ on Heesch number; i.e., suppose every tile that does not admit a tiling has Heesch number less than $H$.

Then we would have an algorithm for checking whether a given tile does not admit a tiling:

Moreover, suppose there is a bound $H$ on Heesch number; i.e., suppose every tile that does not admit a tiling has Heesch number less than $H$.

Then we would have an algorithm for checking whether a given tile does not admit a tiling:

Simply enumerate larger and larger configurations, increasing the number of coronas (shells). If we find a configuration with more than $H$ coronas, we know the tile admits a tiling.

Similarly, suppose there is a bound $I$ on isohedral number; i.e. suppose that every tile that admits a periodic tiling has isohedral number less than $I$.

Similarly, suppose there is a bound $I$ on isohedral number; i.e. suppose that every tile that admits a periodic tiling has isohedral number less than $I$.

Then we would have an algorithm for checking whether a given tile admits a periodic tiling:

Similarly, suppose there is a bound $I$ on isohedral number; i.e. suppose that every tile that admits a periodic tiling has isohedral number less than $I$.

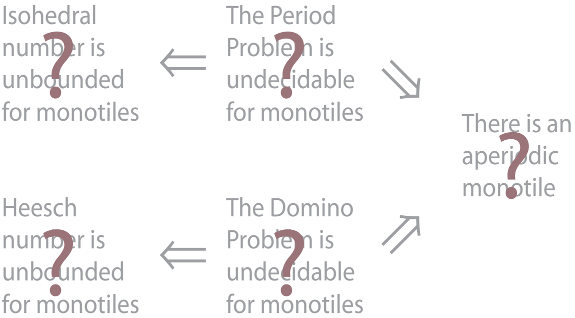
Then we would have an algorithm for checking whether a given tile admits a periodic tiling:

Simply enumerate all configurations with up to $I$ tiles; if we fail to find a fundamental domain, then the tile does not admit a periodic tiling.

And so we have

$$
\begin{tikzcd}
\text{Isohedral number is unbounded for monotiles} & \text{The Period Problem is undecidable for monotiles} \arrow[l, Rightarrow] \arrow[dr, Rightarrow] & \\
& & \text{There is an aperiodic monotile} \\
\text{Heesch number is unbounded for monotiles} & \text{The Domino Problem is undecidable for monotiles} \arrow[l, Rightarrow] \arrow[ur, Rightarrow] &
\end{tikzcd}
$$

And so we have

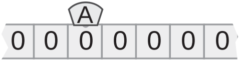



These are all open questions.

## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

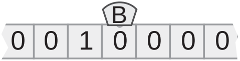
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

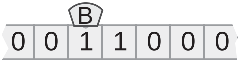
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

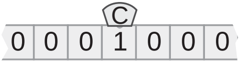
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

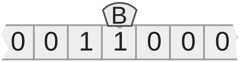
The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |

## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

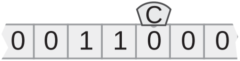
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

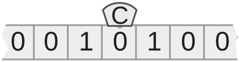
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

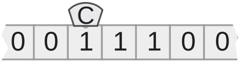
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

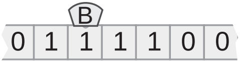
The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |

## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

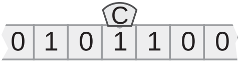
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

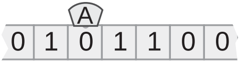
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

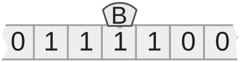
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

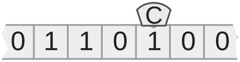
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

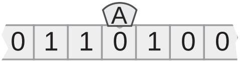
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

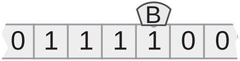
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

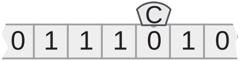
The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |

## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

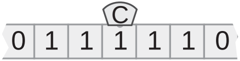
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

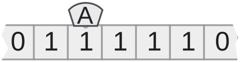
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

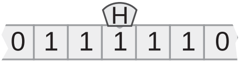
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

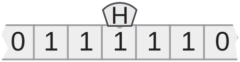
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

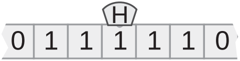
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing’s 1935 construction of a simple, universal model of computation:

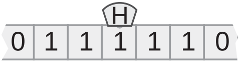
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |



## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |

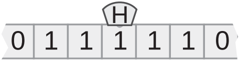



As mysterious as this little example is, it is not too difficult to see how they capture anything we might reasonably mean by "computation";

## Computation and Complexity

The wellspring of all discussions of computational complexity is Alan Turing's 1935 construction of a simple, universal model of computation:

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 1RB | 1LB | 1LC |
| 1 | 1RH | 0RC | 0LA |




As mysterious as this little example is, it is not too difficult to see how they capture anything we might reasonably mean by "computation"; this is the essence of the famous Church-Turing thesis.

And of course the most important undecidable problem is the *Halting Problem*: Does machine $M$ eventually halt?

A simple diagonalization trick shows no machine can take as input $M$ and always halt with the correct answer– the problem is not decidable.

(It’s not hard to discover if a machine halts– just run it until it does– the impossibility is discovering that any given machine will not halt.)

We can mechanically enumerate machines; let $M_n$ be the $n$th machine in any fixed, mechanical enumeration. Then take the following function:

$$H(n) = \begin{cases} \text{time for } M_n \text{ to halt} & \text{if } M_n \text{ does halt} \\ 0 & \text{if } M_n \text{ does not halt} \end{cases}$$

We can mechanically enumerate machines; let $M_n$ be the $n$th machine in any fixed, mechanical enumeration. Then take the following function:

$$H(n) = \begin{cases} \text{time for } M_n \text{ to halt} & \text{if } M_n \text{ does halt} \\ 0 & \text{if } M_n \text{ does not halt} \end{cases}$$

Then *no computable function* can bound $H(n)$.

For suppose some computable $f(n) > H(n)$ for all $n$; then we can calculate whether or not $M_n$ halts by calculating $f(n)$ and running $M_n$ for up to $f(n)$ steps; if $M_n$ hasn't halted by that point, we can be sure it never will. Hence no such $f$ exists.

This all has some interesting consequences.

In particular we can now sketch a proof of Gödel’s theorem, that there are true but unprovable theorems:

In particular we can now sketch a proof of Gödel’s theorem, that there are true but unprovable theorems:

1) Fix a (presumably consistent) formal logical system that captures arithmetic.

In particular we can now sketch a proof of Gödel’s theorem, that there are true but unprovable theorems:

1) Fix a (presumably consistent) formal logical system that captures arithmetic.

Now 2) We can mechanically enumerate all proofs, and hence, all theorems.

In particular we can now sketch a proof of Gödel's theorem, that there are true but unprovable theorems:

1) Fix a (presumably consistent) formal logical system that captures arithmetic.

Now 2) We can mechanically enumerate all proofs, and hence, all theorems.

3) For every program $M$, one or the other of "*M halts*" or "*M does not halt*" is true.

In particular we can now sketch a proof of Gödel's theorem, that there are true but unprovable theorems:

1) Fix a (presumably consistent) formal logical system that captures arithmetic.

Now 2) We can mechanically enumerate all proofs, and hence, all theorems.

3) For every program $M$, one or the other of "*M halts*" or "*M does not halt*" is true.

4) These are statements in our logical system (since running a program, in memory with data, is just arithmetic on giant binary numbers)

In particular we can now sketch a proof of Gödel’s theorem, that there are true but unprovable theorems:

1) Fix a (presumably consistent) formal logical system that captures arithmetic.

Now 2) We can mechanically enumerate all proofs, and hence, all theorems.

3) For every program $M$, one or the other of “*M halts*" or “*M does not halt*" is true.

4) These are statements in our logical system (since running a program, in memory with data, is just arithmetic on giant binary numbers)

But then if every true statement were provable, we’d have a procedure for settling the halting problem: run through all proofs, until you reach one for either “*M halts*" or “*M does not halt*".

In particular we can now sketch a proof of Gödel’s theorem, that there are true but unprovable theorems:

1) Fix a (presumably consistent) formal logical system that captures arithmetic.

Now 2) We can mechanically enumerate all proofs, and hence, all theorems.

3) For every program *M*, one or the other of “*M halts*" or “*M does not halt*" is true.

4) These are statements in our logical system (since running a program, in memory with data, is just arithmetic on giant binary numbers)

But then if every true statement were provable, we’d have a procedure for settling the halting problem: run through all proofs, until you reach one for either “*M halts*" or “*M does not halt*". QED

In fact, if $M$ does halt, then the theorem “*M halts*" can be proven: just run the program! *But the complexity of these theorems cannot be bounded by any computable function!*

In fact, if $M$ does halt, then the theorem "*$M$ halts*" can be proven: just run the program! *But the complexity of these theorems cannot be bounded by any computable function!*

For suppose that the proof of the $n$th theorem of this kind takes no more than some computable $f(n)$ steps. Then again, the Halting Problem would be decidable: given $n$, calculate $f(n)$ and enumerate all proofs up to length $f(n)$. If we find a proof of the theorem, we know $M_n$ halts. On the other hand, if we don't find a proof, we know there'll never be one and $M_n$ does not halt!

With this in hand, we can ask, is our [[whatever]] powerful enough to capture Turing machines?

With this in hand, we can ask, is our [[whatever]] powerful enough to capture Turing machines?

The generic answer is Yes!

**Conway’s Presumption** If enough is going on, your setting is computationally universal.

With this in hand, we can ask, is our [[whatever]] powerful enough to capture Turing machines?

The generic answer is Yes!

**Conway's Presumption** If enough is going on, your setting is computationally universal.

And if if your problem is universal, then one will have

- true but unprovable theorems,
- short theorems with astoundingly lengthy proofs,
- and generally inscrutable behavior.

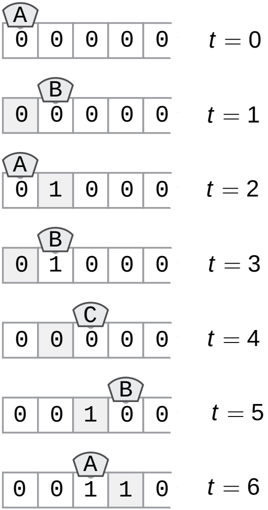
In 1961, Hao Wang gave a simple undecidable tiling problem:

*The Completion Problem: Given a set of tiles and a start configuration, can the configuration be completed to a tiling of the entire plane.*

(Wang was carrying out a larger program of settling the decididability of the remaining cases of Hilbert's "satisfiability" problem: is there an algorithm to decide whether any given first order formula can be satisfied?)

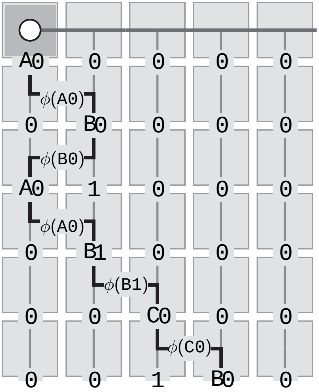
To show the Completion Problem is undecidable, Wang constructed, for any Turing machine, a set of tiles $T$ so that a certain “seed” configuration could be completed to a tiling if and only if the machine fails to halt. Since the Halting Problem is undecidable, so too is the Completion Problem.

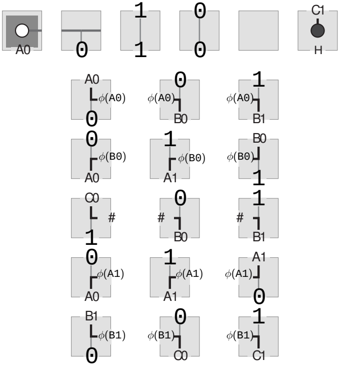
| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 0RB | 1LA | 1RB |
| 1 | 1RB | 0RC | 0LH |



To show the Completion Problem is undecidable, Wang constructed, for any Turing machine, a set of tiles *T* so that a certain "seed" configuration could be completed to a tiling if and only if the machine fails to halt. Since the Halting Problem is undecidable, so too is the Completion Problem.

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 0RB | 1LA | 1RB |
| 1 | 1RB | 0RC | 0LH |






The real point is that Turing machines act in a purely local manner. These tiles perfectly emulate this machine, *in tilings containing the seed tile.*

| $\phi$ | A | B | C |
|---|---|---|---|
| 0 | 0RB | 1LA | 1RB |
| 1 | 1RB | 0RC | 0LH |

Wang noted that if the Domino Problem were undecidable (for sets of tiles) then there would exist aperiodic sets of tiles; he conjectured no such sets exist. In a few years, his student R. Berger proved his conjecture incorrect, and gave the first aperiodic sets of tiles.

Wang noted that if the Domino Problem were undecidable (for sets of tiles) then there would exist aperiodic sets of tiles; he conjectured no such sets exist. In a few years, his student R. Berger proved his conjecture incorrect, and gave the first aperiodic sets of tiles.

We can define the Heesch number of a non-tiling set of tiles to be the maximum radius disk it can cover; just as with the halting times of machine, as we enumerate tile sets, there can be no computable bound on Heesch number.

Wang noted that if the Domino Problem were undecidable (for sets of tiles) then there would exist aperiodic sets of tiles; he conjectured no such sets exist. In a few years, his student R. Berger proved his conjecture incorrect, and gave the first aperiodic sets of tiles.

We can define the Heesch number of a non-tiling set of tiles to be the maximum radius disk it can cover; just as with the halting times of machine, as we enumerate tile sets, there can be no computable bound on Heesch number.

We can define the isohedral number of a periodically tiling set of tiles to be the minimum sized fundamental domain; as we enumerate tile sets, this too can not be bounded by any computable function.

Wang noted that if the Domino Problem were undecidable (for sets of tiles) then there would exist aperiodic sets of tiles; he conjectured no such sets exist. In a few years, his student R. Berger proved his conjecture incorrect, and gave the first aperiodic sets of tiles.

We can define the Heesch number of a non-tiling set of tiles to be the maximum radius disk it can cover; just as with the halting times of machine, as we enumerate tile sets, there can be no computable bound on Heesch number.

We can define the isohedral number of a periodically tiling set of tiles to be the minimum sized fundamental domain; as we enumerate tile sets, this too can not be bounded by any computable function.

Similarly, the minimum length proof that a given set of tiles admits a periodic tiling, or no tiling, cannot be bounded by any computable bound.

Finally, these are asymptotic results, What can one say about the marginal cases? This brings up back to the complexity of our original puzzles. *What can you do with a single tile?*

Finally, these are asymptotic results, What can one say about the marginal cases? This brings up back to the complexity of our original puzzles. *What can you do with a single tile?*

Is tiling by a single tile computationally universal?

Finally, these are asymptotic results, What can one say about the marginal cases? This brings up back to the complexity of our original puzzles. *What can you do with a single tile?*

Is tiling by a single tile computationally universal?

Are Heesch numbers and isohedral numbers unbounded, and if so, do they have computable bounds?

But again, there is nothing intrinsically special about tilings, or Turing machines— Complexity and undecidability are ubiquitous in combinatorial settings (i.e. in all mathematics)!

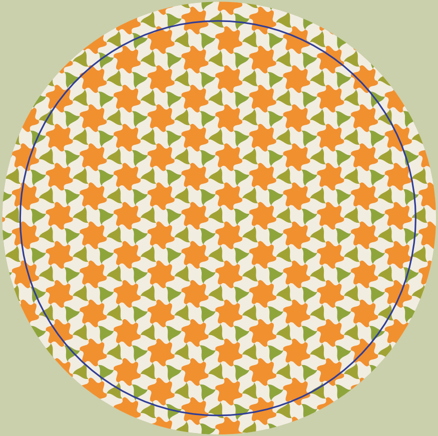
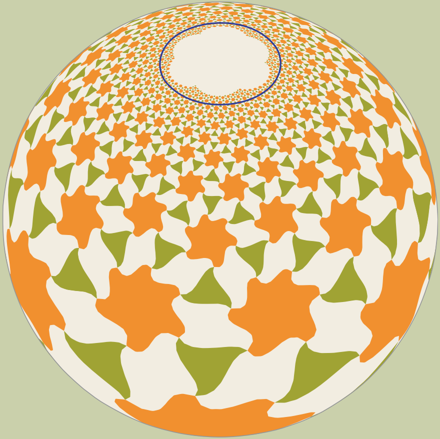
We see this already in simple recreational puzzles!

But again, there is nothing intrinsically special about tilings, or Turing machines— Complexity and undecidability are ubiquitous in combinatorial settings (i.e. in all mathematics)!

We see this already in simple recreational puzzles!

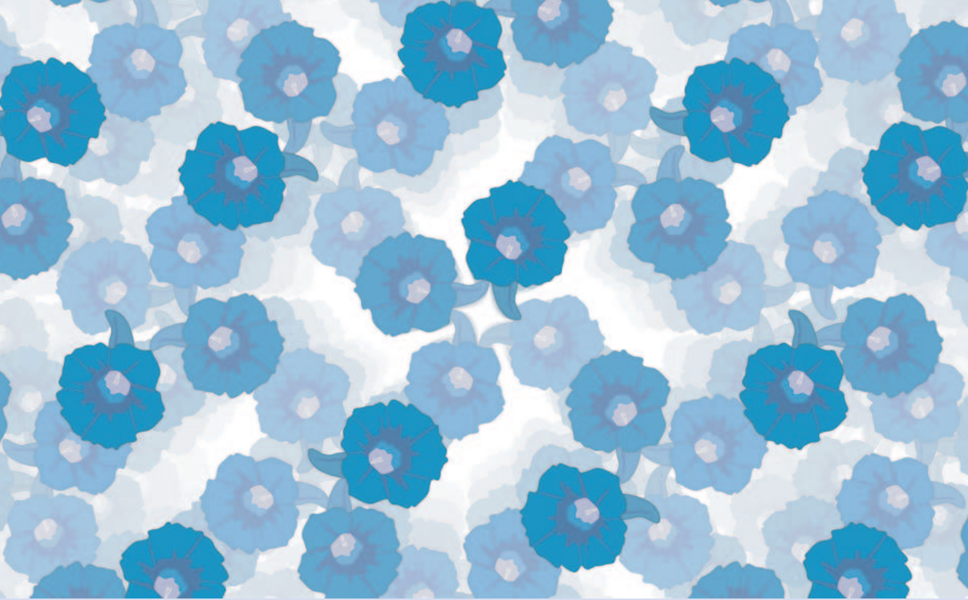
You may keep the toys...

A quick advertisement

# The Symmetries of Things

with John H. Conway & Heidi Burgiel



# The Symmetries of Things

with John H. Conway & Heidi Burgiel

mathbun.com

# The Math Factor

mathfactor.uark.edu