Lecture 9: The Halting Problem

Modelling Complex Systems (Spring 2026)

Jordi-Lluís Figueras

Matematiska Institutionen, Uppsala Universitet

April 22, 2026

Lecture goals

In this lecture we will:

- formulate the halting problem precisely;
- explain why no algorithm can solve the halting problem in complete generality;
- understand the role of self-reference in Turing's argument;
- distinguish between solvable special cases and impossibility in full generality;
- contrast computationally universal cellular automata with the more structured Greenberg-Hastings excitable-medium rule.

Bridge from Lecture 8

Last lecture, the Game of Life showed that a very simple cellular automaton can emulate logic and support Turing-type computation.

This lecture asks the next natural question:

Lecture thesis

Once a system is rich enough to emulate general computation, some global prediction problems become impossible in principle.

We will then contrast that computational richness with the Greenberg-Hastings automaton, whose finite disturbances have a much more structured long-time behavior.

The Halting Problem

What is the halting problem?

The halting problem asks:

> Given a program and an input, will the computation eventually stop, or will it run forever?

This sounds like a very natural question. If we could answer it in general, we would know in advance whether arbitrary computations terminate. For simple programs we often can answer it. The question is whether there exists a single algorithm that always answers correctly for *every* possible program and input.

Warm-up questions

Before the theorem, separate three different tasks:

1. Give one program that obviously halts.
2. Give one program that obviously loops forever.
3. Could there be a single master program that always decides correctly for *every* program and input?

The key distinction is:

- many *individual* instances are easy;
- one *universal* decision procedure is the difficult issue.

A precise formulation

Define

$$\mathrm{HALT}(P, x) = \begin{cases} \text{YES} & \text{if program } P \text{ halts on input } x, \\ \text{NO} & \text{if program } P \text{ runs forever on input } x. \end{cases}$$

We call a program a decider if it always halts and returns a definite answer.

So the question is not whether some cases can be analyzed by insight, but whether there exists a decider for the function $\mathrm{HALT}(P, x)$ in complete generality.

Code as data

A program can be encoded as a finite string. So one program can be given as input to another program.

This means that constructions such as

$$\mathrm{HALT}(P, P)$$

are mathematically legitimate: a program can be tested on its own code.

That is the doorway to self-reference in Turing's argument.

Why this is not just a practical difficulty

The halting problem is not hard merely because programs can be complicated. It is impossible in principle to solve it by a universal algorithm. Turing's theorem says:

Turing's theorem

There is no algorithm that takes as input an arbitrary program together with an arbitrary input and always decides correctly whether that program eventually halts.

So this is a mathematical impossibility result, not a limitation of current technology.

The idea of the proof

The proof is a classical self-reference argument. Assume, for contradiction, that there exists a perfect program `HALT` that always answers:

- `YES` if a given program halts on a given input;
- `NO` if it runs forever.

Then we build a new program that uses `HALT` in a perverse way: it does the opposite of what `HALT` predicts about its own behavior. That construction leads to a contradiction, so the assumed perfect decider `HALT` cannot exist.

The paradox program

Construct a new program `PARADOX(Q)`:

Pseudocode

```
PARADOX(Q):
   if HALT(Q,Q) = YES:
      loop forever
   else:
      halt
```

This program is built to do the opposite of what `HALT` predicts for the self-input pair (Q, Q).

The contradiction step

Now evaluate only one input:

```
PARADOX(PARADOX).
```

If `HALT(PARADOX,PARADOX)` = `YES`	then `PARADOX` loops forever
If `HALT(PARADOX,PARADOX)` = `NO`	then `PARADOX` halts

In both cases the output of `HALT` is false. Therefore a perfect decider for the halting problem cannot exist.

What the theorem does and does not say

The theorem does not say that every individual computation is mysterious. It says that there is no *single general decision procedure* that works for all programs. So there are two different levels:

- for many specific cases, termination can be proved or disproved;
- in full generality, a universal algorithmic test is impossible.

This distinction is exactly what will matter when we compare different cellular automata models.

Misconceptions to avoid

Two questions are worth separating explicitly:

- Does Turing's theorem say that no one can ever prove termination of a specific program? **No.**
- Does it say that there is no single universal algorithm that decides termination for all programs? **Yes.**

The theorem is a limitation on *universal algorithmic prediction*, not on mathematical reasoning about particular systems.

Why this matters for cellular automata

In Lecture 8, the Game of Life served as a prototype of a cellular automaton rich enough to emulate general computation.

So once a cellular automaton reaches Turing-type universality, broad prediction questions about its long-term behavior inherit the same kind of obstruction as the halting problem.

Today we contrast that perspective with the Greenberg-Hastings automaton, whose finite disturbances evolve in a much more structured way.

Greenberg-Hastings Automaton

A simple excitable-medium model

The two-dimensional Greenberg-Hastings cellular automaton is a model on the square lattice $\mathbb{Z}^2$. It was introduced as a simple discrete model for wave patterns in excitable media, such as oscillating chemical reactions. The ingredients are:

- a state set $\{0, 1, \ldots, N-1\}$ with $N \geq 3$;
- an integer $e \in \{1, \ldots, N-2\}$;
- excited states $1, 2, \ldots, e$;
- the four nearest neighbors: north, south, east, and west.

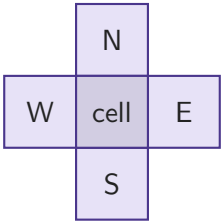
See the phenomenon first

Before focusing on formulas, the main visual features are:

- localized excitation triggers expanding wavefronts;
- recently excited cells enter a refractory stage before returning to rest;
- finite disturbances generate lattice-organized patterns rather than signal-processing logic.

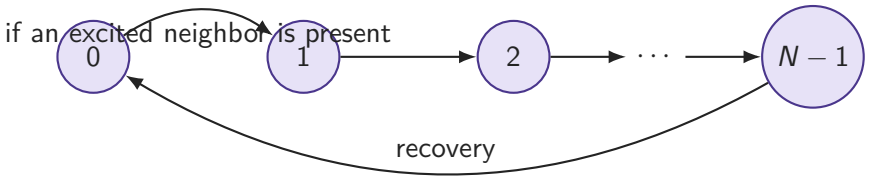
Demo script: `scripts/greenbergHastingsInteractive.py`
Run from the lecture directory with `python3`
`scripts/greenbergHastingsInteractive.py`
For the live demo, start with a single excited cell or a compact seed, then compare that with a random finite patch.

von Neumann neighborhood and local geometry



Only these four neighbors matter. This anisotropy is why large wavefronts on the square lattice look lattice-shaped rather than truly circular.

State cycle



State 0 is resting, states $1, \ldots, e$ are excited, and states $e+1, \ldots, N-1$ are refractory.

Local update rule

At each time step all cells update simultaneously.

- If a cell is in state k with $1 \leq k \leq N-2$, then it advances to $k+1$.
- If a cell is in state $N-1$, then it returns to 0.
- If a cell is in state 0, then:
  - it becomes 1 if at least one of its four neighbors is in an excited state;
  - otherwise it stays at 0.

So state 0 is resting, states 1 through e are excited, and the higher states represent recovery before returning to rest.

Why the dynamics are interesting

Two mechanisms are coupled:

- **reaction:** once a cell is activated, it progresses deterministically through its internal cycle;
- **diffusion:** a resting cell can be triggered by an excited neighbor.

This already produces propagating wavefronts, refractory regions, and large-scale spatio-temporal patterns. The model is therefore a useful example of how a very simple local rule can reproduce qualitative features of excitable media.

Behavior from finite initial data

For initial configurations with only finitely many nonzero cells, the notes state the following qualitative picture:

- every bounded region eventually becomes periodic in time;
- the period may be 1, in which case the activity dies out locally;
- if $N \geq 5$ and $e \geq 2$, then any non-dying structure has temporal period N;
- far from the initial disturbance, the state-1 cells form expanding lattice wavefronts, typically roughly diamond-shaped for the four-neighbor rule, moving outward with speed 1.

So even with deterministic synchronous updating, the model supports organized expanding waves.

Theorem: eventual periodic behavior

Theorem

Assume the initial configuration has only finitely many cells in nonzero states. Then there exists $T \geq 1$ such that for every bounded region $B \subset \mathbb{Z}^2$, the sequence

$$(C_t|_B)_{t \geq 0}$$

is eventually T-periodic.

In addition:

- if $T = 1$, then every bounded region is eventually entirely in state 0;
- if $N \geq 5$ and $e \geq 2$, then every non-dying structure has period exactly N.

Interpretation: local observation windows are eventually periodic even when the full pattern still supports propagating activity.

Simulation versus theorem

The theorem is stated on the infinite lattice $\mathbb{Z}^2$ with finite-support initial data.
In the classroom demo we instead simulate a finite box with zero boundary conditions, so waves eventually interact with the boundary.

Life versus Greenberg-Hastings

Game of Life

- mobile signal-carrying patterns;
- logic gates and Turing-type computation;
- broad prediction inherits computational obstructions.

Greenberg-Hastings

- excitation waves and refractory recovery;
- finite disturbances organize into structured propagation;
- finite-support behavior is much more constrained locally.

Pedagogical point: simple local rules do not all lead to the same level of computational richness or the same limits on predictability.

What should we remember?

- The halting problem asks whether a computation eventually stops.
- Turing proved that no universal algorithm can solve this problem for all programs.
- The proof works by assuming a perfect decider and then forcing it into contradiction.
- The theorem rules out a universal decision procedure, not analysis of every individual case.
- Computational universality brings intrinsic limits on prediction.
- The Greenberg-Hastings automaton is a simple excitable-medium cellular automaton whose finite disturbances evolve in a much more structured way.

Suggested bibliography

For students who want to read more:

- A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, 42(2), 230–265, 1936.
- M. Davis, *Computability and Unsolvability*, Dover, 1982.
- M. Sipser, *Introduction to the Theory of Computation*, Cengage, 2012.
- J. M. Greenberg and S. P. Hastings, "Spatial patterns for discrete models of diffusion in excitable media," *SIAM Journal on Applied Mathematics*, 34(3), 515–523, 1978.

Acknowledgment

These slides were prepared with the assistance of OpenAI Codex / ChatGPT 5.4 as a drafting and editing tool.
All mathematical, pedagogical, and course-specific content remains under the responsibility of the lecturer.