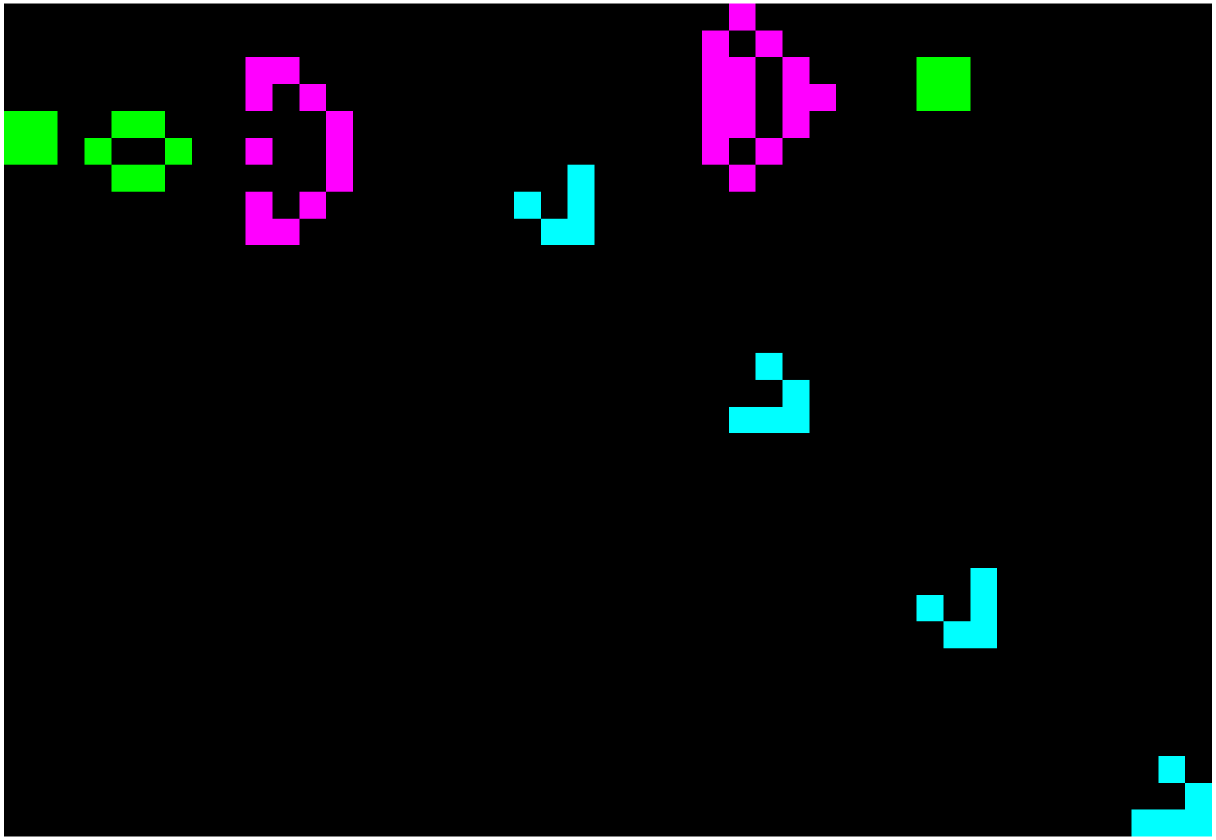
Computability, Undeciability and the Halting Problem

12/01/16

http://www.michael-hogg.co.uk/game_of_life.php



Discrete Structures (CS 173) Lecture B

Gul Agha

Based on slides by Derek Hoiem, University of Illinois

Today's class: Computability, Undecidability and the Halting Problem

- Review of basic concepts from Tuesday
- The halting problem
- Examples of infinitely varying patterns
  - Game of life
  - Penrose tilings

Previous lecture

- Some infinite sets are bigger than others
- A "countable" set is the same size (or smaller) than the natural numbers
- We can compare sizes of infinite sets using bijections or one-to-one functions
- Diagonalization is a useful technique for constructing a new element that is different than every element of a countably infinite set

Size of infinite sets

|Powerset(Naturals)|

||

|Naturals| = |Integers| = |Odd Numbers| < |Reals| < |Powerset(Reals)| < …

Countably Infinite

Functions and Relations on Natural Numbers

$$\mathcal{F} = \{f \mid f: \mathbb{N} \longrightarrow \{0,1\}\}$$

What is the cardinality of $\mathcal{F}$?

Notice that this encodes the power set of natural numbers!

Uncomputability

- A computer program is just a string (finite series of characters), so the set of all programs is countable.
- But the set of functions is uncountable (e.g., functions that map reals to reals)
- So there are more functions than programs – some functions cannot be computed by any program

Cardinality of the Set of all Programs

Programs are from a countable grammar (rules, kinds of statements) and of finite (though arbitrary) length. How many programs may have:

- “Length” 1
- “Length” 2
- …

Example: not all real numbers are computable

3/4

$\sqrt{2}$

http://keisan.casio.com/calculator

Halting problem:

- Is there a general purpose algorithm that can determine whether a program will terminate?
  - On a given input?
  - On all inputs?

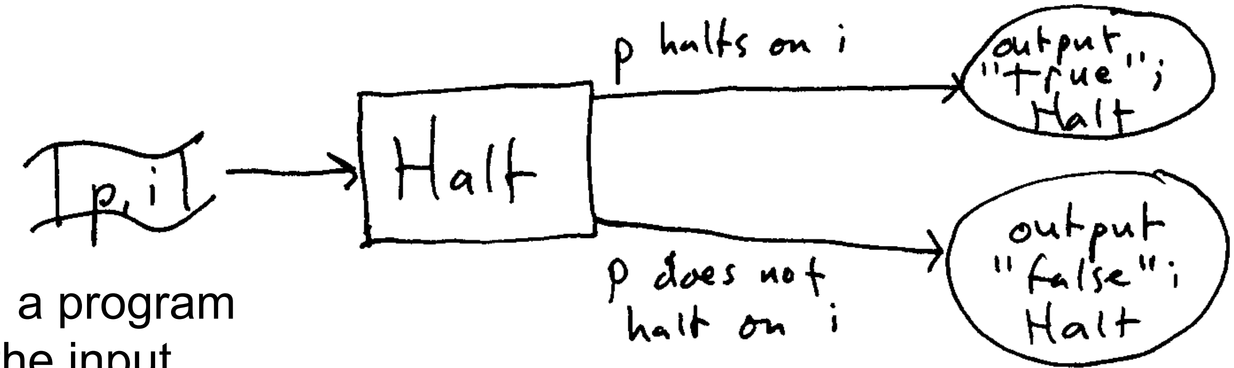
Example: Suppose you have written a machine learning algorithm that has been running for days. Will it ever stop on its own?

Potential solutions to halting problem

- Run program for a really long time and see if it stops.
- Analyze code to see if there are infinite loops.
- Check if loop exit conditions become “closer” to being met over time (Induction proofs).

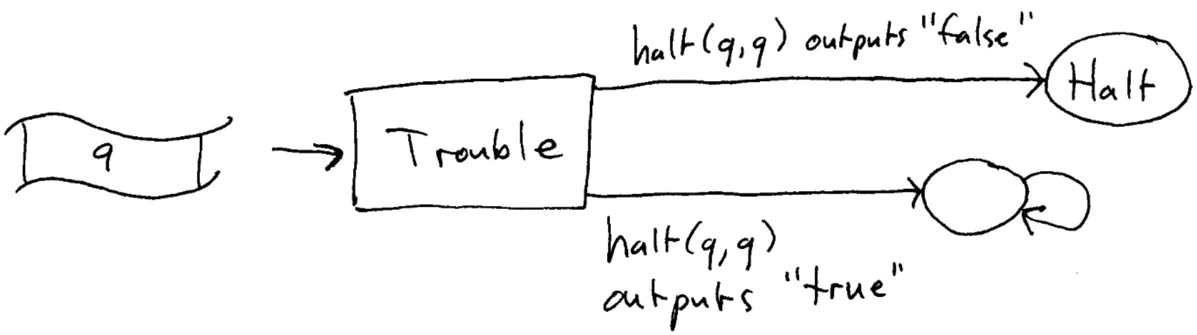
Halting problem is undecidable

Suppose we have a magic program that solves the halting function:



p = a program
i = the input

Now we write this evil program:



What happens if we call trouble(trouble)?

Example source: http://faculty.ycp.edu/~dhovemey/fall2009/cs340/lecture/lecture22.html

Liar’s paradox

- In some primitive kingdom, they had not abolished the death penalty. A King ruled that his critics should be executed. But to torture them further, he decreed that they could make a statement. If the statement is true, they should be beheaded. If the statement is false, they should be hanged.
- One of his critics was a logician. She said: “I will be hanged.”
- True or False: “This statement is false”

Halting problem is undecidable

```
bool does_it_halt( char * program, char * input ) {
  if( some terribly clever test for halting )
    { return TRUE; }
  else
    { return FALSE; }
}

bool evil_program( char * program ) {
  if( does_it_halt( program, program ) )
    { while( 1 ) {} return FALSE; }
  else
    { return TRUE; }
}
```

What does `evil_program(evil_program)`do?

Turing's Proof.   Example from: http://www.cgl.uwaterloo.ca/~csk/halt/

Why the halting problem matters

- We often want to know if a program converges (halts), but not possible to provide one algorithm that answers this for all programs.
- In practice, convergence can be proven in many cases
- First example of a decidability problem
  - Many other problems are shown to be undecidable by being reduced to halting problem
  - E.g., there cannot be a general algorithm that decides whether a given statement about natural numbers is true or not

Halting behaviors

- Program halts
- Program loops but keeps repeating itself
  - Sometimes detectable, unless loop is really long
- Program continues changing without repetition
  - Hard to tell if it will stop or start repeating or keep changing forever

Example: is there a finite or repeating set of digits?

3/4

1/7

102/103

$\sqrt{2}$

http://keisan.casio.com/calculator

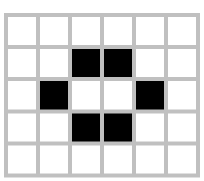
Another example: game of life

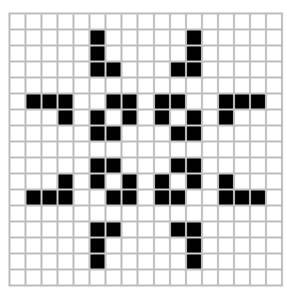
Simple rules

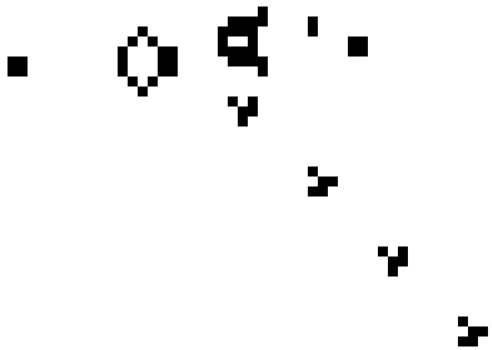
- Any live cell with fewer than two live neighbours dies, as if caused by under-population.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overcrowding.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Complex outputs

- Some programs halt, some loop repeatedly, some continually change







Historical significance of Conway's Game of Life

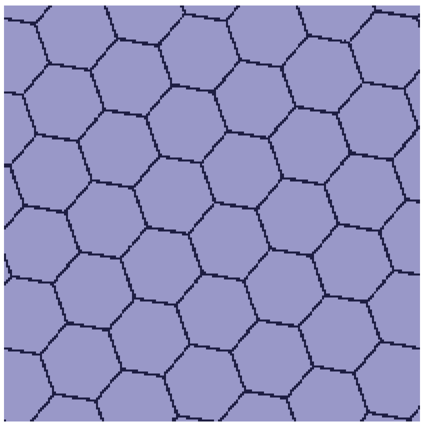
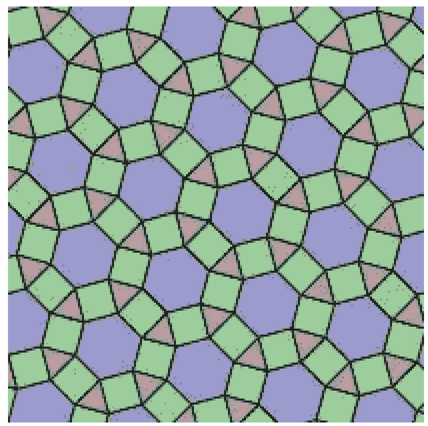
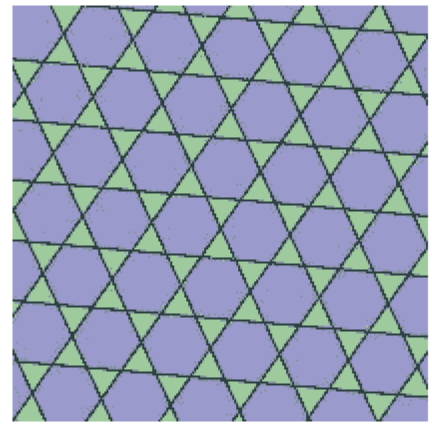
- Proposed in answer to question of whether self-replicating machines are possible – artificial life
  - Invented in 1970 (on paper)
- Example of emergence and self-organization

Game of life online:
http://www.bitstorm.org/gameoflife/
http://www.ibiblio.org/lifepatterns/

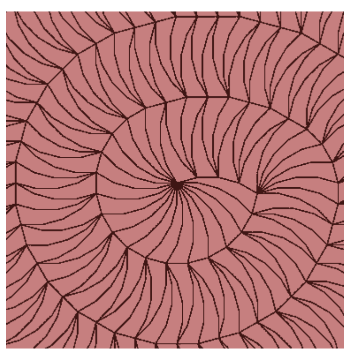
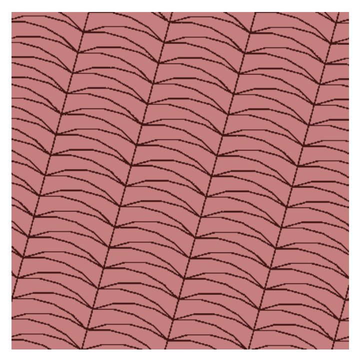
Aperiodic tilings

Examples from http://www.ams.org/samplings/feature-column/fcarc-penrose

Tiles that lead to periodic tilings

Tiles that can create periodic or aperiodic tilings

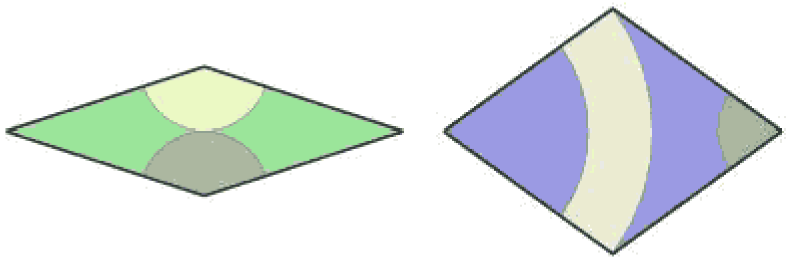
Question asked by Hao Wang in 1961:

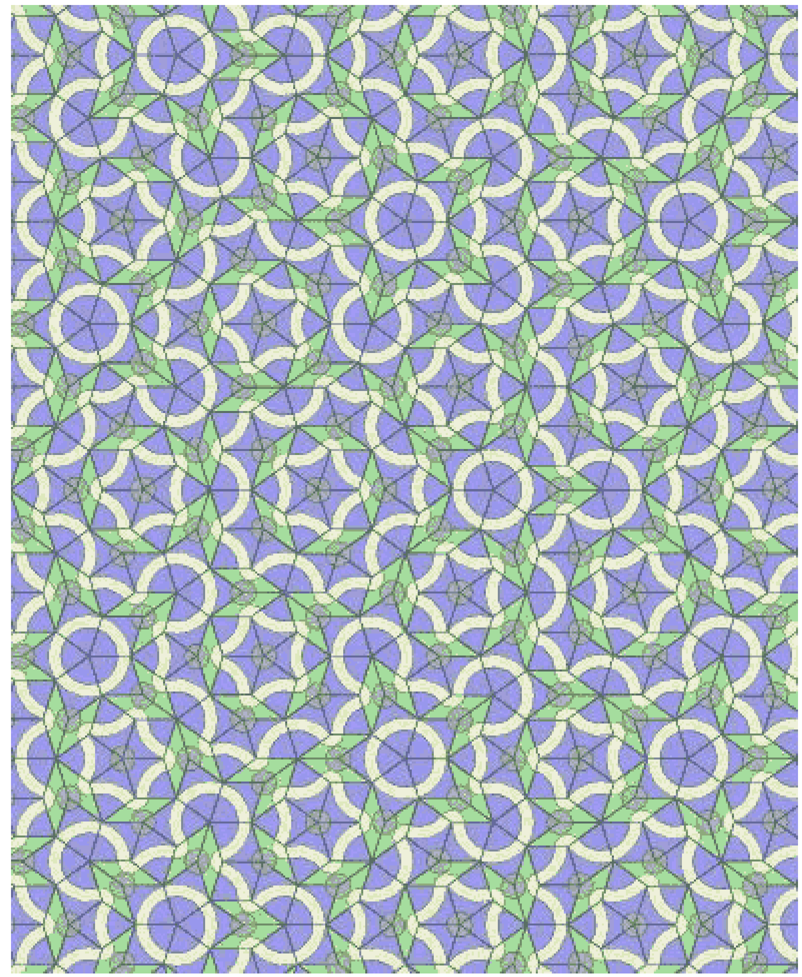
Is it possible to create tiles that always lead to aperiodic tilings?

First solved by his student Berger in 1963 with 20,000+ tiles

Is it possible to create tiles that always lead to aperiodic tilings?

Penrose tilings

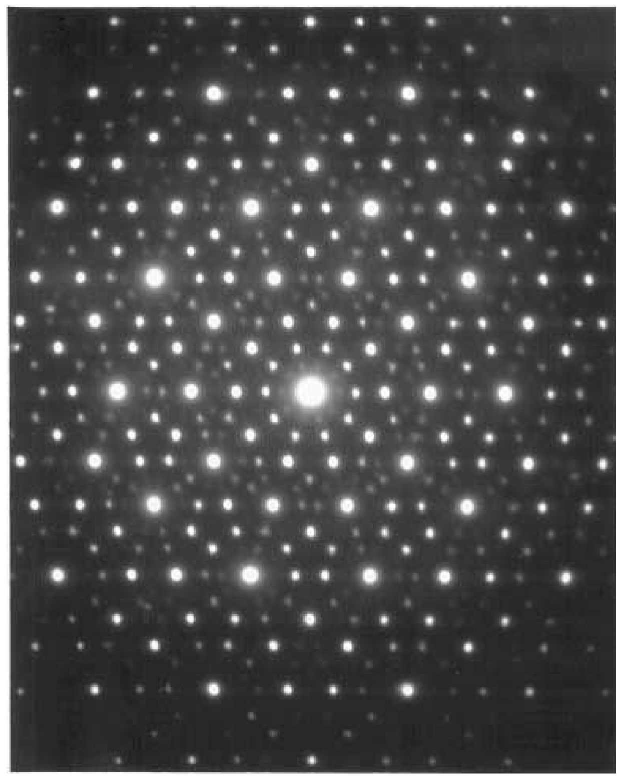




Examples from
http://www.ams.org/samplings/feature-column/fcarc-penrose

Application: quasicrystals

- Eventually understood to be analogous to 3D Penrose tilings



More on this here: http://en.wikipedia.org/wiki/Quasicrystal

Image from Ron Lifschitz

Universal Model of Computing

Church's Thesis: Anything that is computable can be computed by a *Turing Machine.*

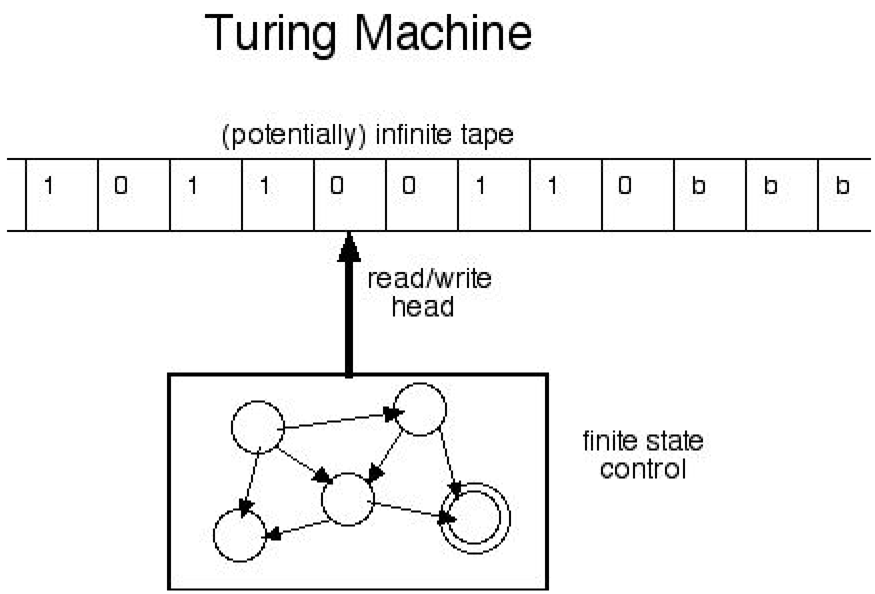
A model or programming language is *Turing Complete* if it can compute anything a TM can.

Examples:

1. *the λ-calculus* (theory of functions)
2. *C, C++, algol, Java, Scheme, …*

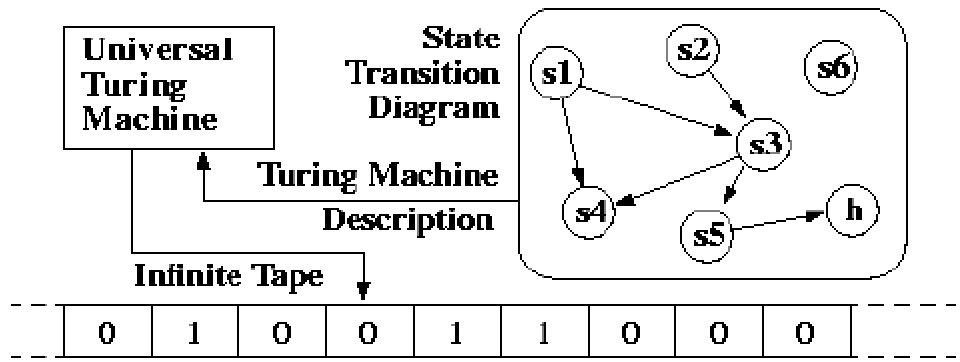
What is a Turing Machine?

Turing Machines



https://csustan.csustan.edu/~tom/Lecture-Notes/Computation/Turing-Machine.jpg

A description of the Finite State automata for control can be entered on the tape to be interpreted by a Turing Machine (called a Universal Turing Machine because it can execute an arbirtrary given TM on a given input)



http://web.mit.edu/manoli/turing/www/turing.gif

See also: https://en.wikipedia.org/wiki/Turing_machine

Some Undecidable Problems

- Given an arbitrary function f:
  - Is f a *total function?*
  - Is $f(0)$ *defined?*
  - Is there a natural number for which $f(n)$ is defined?
- The Equivalence Problem
- Post's Correspondence Problem
- Hilbert's Tenth Problem

The Equivalence Problem

- Given two (arbitrary) computable functions (programs) is there an algorithm that can decide if the functions produce the same output?

  Example: $f(x) = x + x \qquad g(x) = 2x$

- *Simpler version:* given an arbitrary computable function, is there an algorithm to decide if it is equivalent to the *identity* function?
- *Neither is decidable.*
- *But if two computable functions are not equivalent, you can find that they are not.*

Post's Correspondence Problem

- Introduced by Post in 1946.
- Best illustrated with an instance. Given a finite sequence of pairs of strings:

$$(s_1, t_1), (s_2, t_2), \dots, (s_\mathrm{n}, t_\mathrm{n})$$

Is there a sequence of indices $i_1, \dots, i_k$, with repetitions allowed, such that:

$$s_{i_1} \dots s_{i_k} = t_{i_1} \dots t_{i_k}$$

Two instances:

$$(ab, a), (b, bb), (aa, b), (b, aab)$$

A solution: sequence 1, 2, 1, 3, 4 gives:

$$(ab, a), (b, ab)$$

Hilbert's Tenth Problem

Does a polynomial $p(x_1, \dots, x_n) = 0$ with integer coefficients have a solution such that each of $x_1, \dots, x_n$ are integers?

(Posed by Hilbert in 1900).

Some instances can be solved:

$$x_1 + x_2 = 2$$

$$x^2 = 2$$

In 1970, Matiyasevich proved that Hilbert's Tenth Problem is undecidable. (There is no algorithm that can say if a polynomial has integer solutions).