Seminar : Automata Theory

Lec 3 : Further Topics

Jinhee Paeng

Sep 7, 2023

Seoul National University

Table of Contents

Table of Contents

P vs NP

Remark. In this section we will cover *complexity classes*:

- P
- NP, co-NP
- NP-complete
- NP-Hard

class P

Definition (P)

A language L is in P if and only if there exists a deterministic Turing Machine M, such that

- M runs for polynomial time on all inputs.
- For all x in L, M outputs y.
- For all x not in L, M outputs n.

Remark. Class P is what we usually call *easy* problems.

class NP

Definition (NP)

A language L is in P if and only if there exists a nondeterministic Turing Machine M, such that

- M runs for polynomial time on all inputs.
- For all x in L, there's a path where M outputs y.
- For all x not in L, every path of M outputs n.

Remark. Class NP is what we usually call *hard* problems.

Remark. We had not define *nondeterministic TM*, but it is the same extension as *DFA* to *NFA*.

class NP

Definition (NP - verifier definition)

A language L is in NP if and only if there exists a deterministic Turing Machine V, such that

- V runs for polynomial time on all inputs of ordered pair (I, W).
- W is the question we want to answer, "is $W \in L$?"
- I is the instance that we want to be a proof of the answer.
- If I proves $W \in L$, then V outputs y.
- If I cannot prove $W \in L$, then V outputs n.

Example. Let L be the question of whether given set has a subset of sum zero. If an example of sum zero is given, then it is easy to check that it has a subset of sum zero.

class NP

Theorem (Equivalence of definitions)

Both definitions of NP is identical.

Proof.

- If a verifier is provided, generate all instances by appending an alphabet, run each at the verifier.
- If a NTM is provided, there exists an instance that reaches the y state in polynomial time. Use that instance.

□

Definition (co-NP)

A language L is in *co-NP* if L^C is *NP*.

Remark. We can also define similarly as of *NP*:

- NTM: there exists a path to n if $x \notin L$.
- Verifier: there's a verifier that can prove $x \notin L$ with a counter-example.

Open question. *NP vs co-NP*. Are they same?

Remark. $P \subset NP \cap co\text{-}NP$

P vs NP

The statement of millennial problem "P vs NP" is quite simple.

P vs NP.

Is the complexity class P and NP are identical?

Remark. It is obvious that $P \subset NP$. The question is the converse.

Remark. We can interpret this as whether currently "hard" problems has "easy" solution. However, there's currently no answer to it.

Remark. However, there's some proofs that certain types of proofs such as "natural proof system" cannot prove or disprove this problem.

NP-hard

Definition (NP-hard)

A decision problem C is *NP-hard* if Every problem in *NP* is reducible to C in polynomial time.

Remark. It was proven that *Super Mario Bros.* is *NP-hard*.

Super Mario Bros. Is Harder/Easier than We Thought

Erik D. Demaine[1], Giovanni Viglietta[2], and Aaron Williams[3]

1 Computer Science and Artificial Intelligence Laboratory, MIT, 32 Vassar St., Cambridge, MA 02139, USA
edemaine@mit.edu
2 School of Electrical Engineering and Computer Science, University of Ottawa, Canada
viglietta@gmail.com
3 Division of Science, Mathematics, and Computing, Bard College at Simon's Rock, 84 Alford Rd, Great Barrington, MA 01230, USA
haron@uvic.ca

Abstract

Mario is back! In this sequel, we prove that solving a generalized level of Super Mario Bros. is PSPACE-complete, strengthening the previous NP-hardness result (FUN 2014). Both our PSPACE-hardness and the previous NP-hardness use levels of arbitrary dimensions and require either arbitrarily large screens or a game engine that remembers the state of off-screen sprites. We also analyze the complexity of the less general case where the screen size is constant, the number of on-screen sprites is constant, and the game engine forgets the state of everything substantially off-screen, as in most, if not all, Super Mario Bros. video games. In this case we prove that the game is solvable in polynomial time, assuming levels are explicitly encoded; on the other hand, if levels can be represented using run-length encoding, then the problem is weakly NP-hard (even if levels have only constant height, as in the video games). All of our hardness proofs are also resilient to known glitches in Super Mario Bros., unlike the previous NP-hardness proof.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases video games, computational complexity, PSPACE

Digital Object Identifier 10.4230/LIPIcs.FUN.2016.13

NP-complete

Definition (NP-complete)

A decision problem C is *NP-complete* if it is both *NP* and *NP-hard*.

Remark. *NP-complete* problem is important since

- developing polynomial time solver proves $P = NP$.
- proving non-existence ot polynomial time solver disproves $P = NP$.

Example.

- Boolean satisfiability problem (SAT)
- Knapsack problem
- Travelling salesman problem (decision version)
- N-Queen problem
- etc..

SAT problem

Definition

A boolean expression consisted with *And, Or, Not, $(\cdot)$* is called *satisfiable* if there's an instance that makes the output *true*.
The *SAT* problem is to determine whether given expression is *satisfiable*.

SAT is NP-complete.

For a given *NTM* that solves a *NP* problem, assume it runs for polynomial time $p(n)$ for an input size n. Given an input, check the input size n and construct a boolean function with variables of:

Variables	Intended interpretation	How many?[7]
$T_{i,j,k}$	True if tape cell i contains symbol j at step k of the computation.	$O(p(n)^2)$
$H_{i,k}$	True if M's read/write head is at tape cell i at step k of the computation.	$O(p(n)^2)$
$Q_{q,k}$	True if M is in state q at step k of the computation.	$O(p(n))$

SAT problem

SAT is NP-complete, continued.

and the function as a conjugation (*And*, multiplication operator) of:

Expression	Conditions	Interpretation	How many?
$T_{i,j,0}$	Tape cell i initially contains symbol j	Initial contents of the tape. For $i > n-1$ and $i < 0$, outside of the actual input I, the initial symbol is the special default/blank symbol.	$O(p(n))$
$Q_{s,0}$		Initial state of M.	1
$H_{0,0}$		Initial position of read/write head.	1
$\neg T_{i,j,k} \lor \neg T_{i,j',k}$	$j \neq j'$	At most one symbol per tape cell.	$O(p(n)^2)$
$\bigvee_{j \in \Sigma} T_{i,j,k}$		At least one symbol per tape cell.	$O(p(n)^2)$
$T_{i,j,k} \land T_{i,j',k+1} \to H_{i,k}$	$j \neq j'$	Tape remains unchanged unless written by head.	$O(p(n)^2)$
$\neg Q_{q,k} \lor \neg Q_{q',k}$	$q \neq q'$	Only one state at a time.	$O(p(n))$
$\neg H_{i,k} \lor \neg H_{i',k}$	$i \neq i'$	Only one head position at a time.	$O(p(n)^3)$
$(H_{i,k} \land Q_{q,k} \land T_{i,\sigma,k}) \to \bigvee_{((q,\sigma),(q',\sigma',d)) \in \delta} (H_{i+d,\, k+1} \land Q_{q',\, k+1} \land T_{i,\, \sigma',\, k+1})$	$k < p(n)$	Possible transitions at computation step k when head is at position i.	$O(p(n)^2)$
$\bigvee_{0 \le k \le p(n)} \bigvee_{f \in F} Q_{f,k}$		Must finish in an accepting state, not later than in step $p(n)$.	1

Computation of such polynomial results can be done in polynomial time. Thus, *SAT* is NP-hard. *SAT* being NP is quite trivial. □

Remark. This result is called *Cook–Levin theorem*.

Remark. Proof of *NP-Completeness* uses this result of *SAT* problem and perform a reduction.

Table of Contents

Hilbert's 10th Problem

Hilbert's 10th Problem: The original statement is

"Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers."

Hilbert's 10th Problem

In our words,

Definition (Hilbert's 10th Problem)

Is there an algorithm (Turing Machine) that decides whether a given Diophantine equation with integer coefficients has a solution in integers?

However, it is proven to be impossible.

Theorem (Y.Matiyasevich, 1970)

Hilbert 10th Problem is negative:
It is impossible to build a Turing Machine that decides whether given Diophantine equation with integer coefficients has a solution in integers.

Goal. The goal of this section is to see how *Undecidablility* is applied.

Diophantine set

Definition (Parameters, Variables)

Consider a Diophantine equation with integer coefficients :

$$D(a_1, a_2, \cdots, a_r, x_1, x_2, \cdots, x_n) = 0.$$

We call $a_1, a_2, \cdots, a_r$ as *parameters* and $x_1, x_2, \cdots x_n$ as *variables*. Note that there's no difference between parameters and variables.

Definition (Diophantine set)

A set $A \subset \mathbb{Z}^r$ is called *Diophantine* if there exists a *Diophantine equation* D such that

$$A = \left\{(a_1, a_2, \cdots, a_r) : \exists (x_1, x_2, \cdots, x_n) \text{ s.t. } D(a, x) = 0\right\}.$$

Lagrange's four-square theorem

Example. Set of natural numbers $\mathbb{N}$ is an example of *Diophantine set*.

$$D(a, x_1, x_2, x_3, x_4) = x_1^2 + x_2^2 + x_3^2 + x_4^2 - a + 1$$

Theorem (Lagrange's four-square theorem)

For any natural number n, there exist four integers x_1, x_2, x_3, x_4 such that

$$n = x_1^2 + x_2^2 + x_3^2 + x_4^2.$$

Recursively Enumerable set

Definition (Recursively Enumerable set)

A set $A \subset \mathbb{Z}^r$ is called *Recursively Enumerable* if there exists a Turing Machine that prints each element in A eventually.

Remark. As one can see from the name, such set is equivalent to *Recursively Enumerable language*. If there exists a TM M that defines A, consider a new TM that

1. Consider $k = 10$.
2. Run M for k iterations for all inputs $a < k$. If it halts, $a \in A$.
3. $k \leftarrow 10k$ and run from *step 2* again.

Recursive set

Definition (Recursive set)

A set $A \subset \mathbb{Z}^r$ is called *Recursive* if there exists a Turing Machine that decides A, i.e. halts with state *yes* if $a \in A$, halts with state *no* if $a \notin A$.

Example.

- Set of prime numbers.
- Set of even numbers.
- etc...

RE but not Recursive

Theorem

Recursive set is always Recursively Enumerable set.

Remark. However, the converse does not hold.

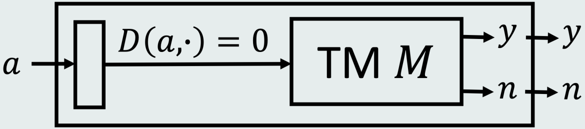
Example. Consider a set of natural numbers defined as

$$\{2^p 3^x : p = \langle \omega \rangle, x = \langle M \rangle, \omega \in L(M)\}.$$

Remark. For the reminder, *Halting problem* was an instance of *Recursively Enumerable* but not *Recursive Language*.

Proof of Hilbert's 10th Problem.

Suppose there exists a TM M that decides the existence of the solution. Then, every *Diophantine set* is *Recursive set*. Assume *Diophantine set* A and its corresponding Diophantine equation $D(a, x) = 0$ is given.



Then, the TM above decides the A making it *Recursive set*.
However, it is now proven that a set is *Diophantine set* if and if only *Recursively Enumerable set*. This yields a contradiction on *Recursively Enumerable set* is not always *Recursive*.

Diophantine = RE?

Question. Now, all we have to answer is whether *Recursively Enumerable set* is equivalent to *Diophantine*.

Ideas of the proof.

To do this, Davis, Putnam, Robinson, Matiyasevich built the Turing Machine with the *Diophantine* equation. They built a *Diophantine* equation

$$D(p, t, k) = 0,$$

- p represents the current state and the location of the head.
- t represents the current tape.
- k represents the iteration number.

where the equation holds when current configuration (p, t) halts after k iterations. This is done by proving transfer function is *Diophatine*. $\square$

Diophantine = RE

Reduction. The problem can be reduced to *non-negative solution*.

- If $\exists$algorithm that decides $\exists$solutions in $\mathbb{N}$, it also does in $\mathbb{Z}$.

Proof.

To solve $D(z_1, z_2, \cdots, z_n) = 0$ in $z \in \mathbb{Z}^n$, then solve
$D(x_1 - y_1, x_2 - y_2, \cdots, x_n - y_n) = 0$ for $x, y \in \mathbb{N}^n$. $\square$

- If $\exists$algorithm that decides $\exists$solutions in $\mathbb{Z}$, it also does in $\mathbb{N}$.

Proof.

To solve $D(x_1, x_2, \cdots, x_n) = 0$ in $x \in \mathbb{Z}^n$, then solve
$D(a_1^2 + b_1 + c_1^2 + d_1^2, \cdots, a_n^2 + b_n + c_n^2 + d_n^2) = 0$ for $a, b, c, d \in \mathbb{Z}^n$. $\square$

Diophantine = RE

Idea 1. *Diophantine* equation can do *And* operator $\cap$.

Proof.

Consider a *Diophantine* equation:

$$D_1^2 + D_2^2 = 0.$$

$\square$

Idea 2. *Diophantine* equation can do *Or* operator $\cup$.

Proof.

Consider a *Diophantine* equation:

$$D_1 \times D_2 = 0.$$

$\square$

Diophantine = RE

Example 1. A relation $a \leq b$ is *Diophantine*:

$$D(a, b, x) := b - a - x = 0, \quad x \in \mathbb{N}.$$

Example 2. A relation $a|b$ is *Diophantine*:

$$D(a, b, x) := b - ax = 0, \quad x \in \mathbb{N}.$$

Example 3. A relation $a \equiv b \mod c$ is *Diophantine*:

$$D(a, b, c, x) = b - a - cx = 0, \quad x \in \mathbb{N}.$$

Diophantine = RE

Example 4. A set of composite numbers is *Diophantine*:

$$D(t, x, y) := t - (x + 2)(y + 2) = 0, \quad x, y \in \mathbb{N}.$$

Example 5. A set of non-powers of 2 is *Diophantine*:

$$D(t, x, y) := t - (2x + 3)y = 0, \quad x, y \in \mathbb{N}.$$

Remark. The complement of the example 4,5 are also *Recursively Enumerable*. However, there corresponding equations are very complex.

Diophantine = RE

Corollary 1. There exists a *Diophatine* $D(p, x)$, where

- $D = 0$ only when p is a prime number.
- when p is prime, there exist a tuple of integers x that makes $D = 0$.

Corollary 2. There exists a *Diophatine* that the positive elements of the range set is the set of prime numbers.

$$P_D(t, x) = (t + 1)(1 - D(t, x)^2) - 1$$

This polynomial returns a value t when $D(t, x) = 0$, and a negative value when $D(t, x) \neq 0$.

Diophantine = RE

Jones et al. (1976) found the explicit form:

$$(k+2) \text{ is prime} \Leftrightarrow \text{solution exists for } (k+2)(1-\sum_i \alpha_i^2) > 0,$$

$$\alpha_0 = wz + h + j - q = 0$$
$$\alpha_1 = (gk + 2g + k + 1)(h + j) + h - z = 0$$
$$\alpha_2 = 16(k+1)^3(k+2)(n+1)^2 + 1 - f^2 = 0$$
$$\alpha_3 = 2n + p + q + z - e = 0$$
$$\alpha_4 = e^3(e+2)(a+1)^2 + 1 - o^2 = 0$$
$$\alpha_5 = (a^2-1)y^2 + 1 - x^2 = 0$$
$$\alpha_6 = 16r^2y^4(a^2-1) + 1 - u^2 = 0$$
$$\alpha_7 = n + \ell + v - y = 0$$
$$\alpha_8 = (a^2-1)\ell^2 + 1 - m^2 = 0$$
$$\alpha_9 = ai + k + 1 - \ell - i = 0$$
$$\alpha_{10} = ((a + u^2(u^2-a))^2 - 1)(n + 4dy)^2 + 1 - (x + cu)^2 = 0$$
$$\alpha_{11} = p + \ell(a - n - 1) + b(2an + 2a - n^2 - 2n - 2) - m = 0$$
$$\alpha_{12} = q + y(a - p - 1) + s(2ap + 2a - p^2 - 2p - 2) - x = 0$$
$$\alpha_{13} = z + p\ell(a - p) + t(2ap - p^2 - 1) - pm = 0$$

Diophantine = RE

Some histories:

1. "Diophantine = RE" is conjectured by *Davis (1949)*.
2. Solved with *exponential Diophantine* (ex. $2x^{3y^x+z}$) by *Davis, Putnam, Robinson (1959)*. Thus, it makes proving

   $$\{(a, b, c) \in \mathbb{N}^3 : c = a^b\}$$

   is *Diophantine* proves $H10$.
3. *Matiyasevich (1970)* proves the set

   $$\{(a, b) \in \mathbb{N}^2 : b = F_{2a}\}$$

   is *Diophatine* and concludes the $H10$.