Playing Games for "Research Purposes"

Complexity Analysis

Team AN

“Gaming is Hard, but someone must ~~do~~ prove it.”

– A plagiarised quote by Kunwar

Our main focus

To classify CELESTE into a complexity class, to study Complexity Theory in depth and discuss the inherent connection between Games and Complexity Theory.

What we have accomplished

- A solid understanding of Complexity Theory.
- Proved the video game 'Celeste' to be NP-complete.
- With a few additions, proved the level to be PSPACE-Complete.
- Studied how Game Theory ties with Complexity Theory.
- Studied other formalisms related to modelling games
- Presented Generalizations about Games in general as well as 2D-Platformers in particular

Disclaimer: Whenever we say "Celeste belongs to X complexity class", we mean to say that the decision problem of deciding whether finish point is reachable from start point.

So what is Complexity Theory?

- Complexity Theory is the study of complexity and of complex systems. It helps computer scientists relate and group problems together into complexity classes.

- Sometimes, if one problem can be solved, it opens a way to solve other problems in its complexity class.

- Complexity helps determine the difficulty of a problem, often measured by how much time and space (memory) it takes to solve a particular problem.

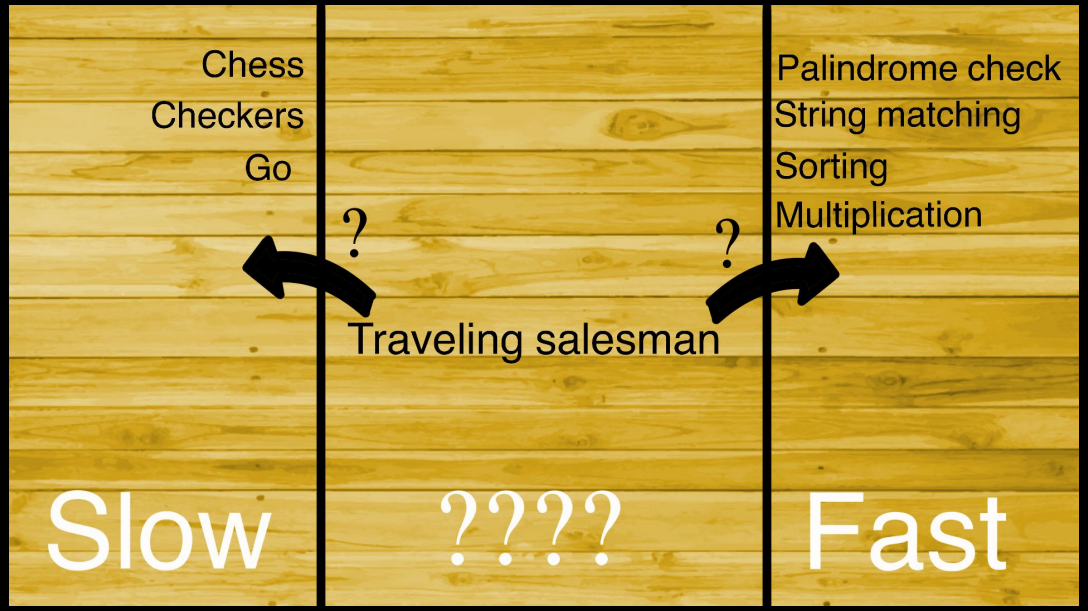
- Both in theory and in practice, complexity theory helps computer scientists determine the limits of what computers can and cannot do.

- Complexity Class contains a set of problems which take a similar range of space and time to solve.

- Problems are usually proven to be in a particular complexity class by running the problem on an abstract computational model, usually a Turing Machine.

The Complexity Zoo

The need of classification



Is there something particular that makes the problems hard?

How do we compare different models of computation?

Hierarchy

One of the facts leading to the hierarchy of the models and problems is that more resources result in more computational capacity.

Theorems that aided in forming a hierarchy:

- Time Hierarchy theorem
- Space Hierarchy theorem
- Savitch's theorem

Models and variants covered:

- Deterministic and nondeterministic Turing Machines
- Probabilistic Turing Machines
- Quantum computers

Relations between classes

Recognizing puzzles harder than solving them? P vs NP

Conjectures which when solved will solve this P vs NP.
For example, the Berman-Hartmanis Conjecture.

"If P=NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in 'creative leaps,' no fundamental gap between solving a problem and recognizing the solution once it's found."

- Scott Aaronson

"One day I will find the right words, and they will be simple."

-Jack Kerouac

Adding randomization to computation

Solovay and Strassen idea of making an algorithm for primality check with the help of coin-flipping. Much better algorithms now, but the idea was the key.

Randomization means not correct answer always, accuracy becomes another measure of performance, resulting in more subclasses.

Answer \ Output	1	0
1	$\geq 1/2$	$\leq 1/2$
0	0	1

Running algorithms which can give incorrect answer for polynomial time: Monte Carlo

Complete Complexity zoo can be covered

Complexity Class is very big, one can not read it completely. We studied the general structure of the complexity zoo and how the hierarchy is.

We have enough understanding about the complexity zoo to learn about any other class or any other new computational model.

As Grothendieck once taught us, objects aren't of great importance; It is the relationship between them that are.

The Cook Levin Theorem

Cook Levin Theorem (also known as Cook's Theorem) states that the Boolean Satisfiability Problem(SAT) is NP-complete.

An important consequence of this theorem is that if there exists a deterministic polynomial time algorithm for solving Boolean satisfiability, then every NP problem can be solved by a deterministic polynomial time algorithm.

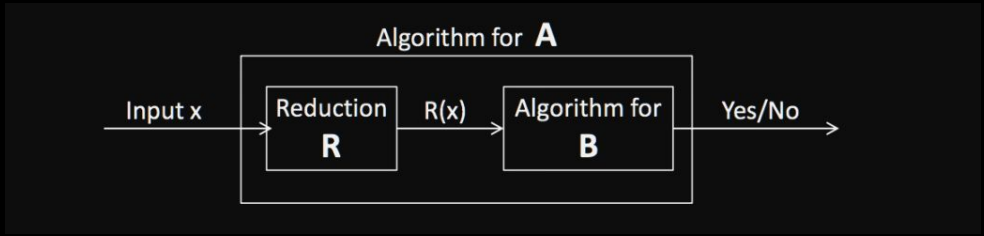
The question of whether such an algorithm for Boolean satisfiability exists is thus equivalent to the P v/s NP problem, which is widely considered the most important unsolved problem in theoretical computer science.

Karp's Reduction

A polynomial-time algorithm for transforming inputs to one problem into inputs to another problem, such that the transformed problem has the same output as the original.

Let $A : X \longrightarrow \{0, 1\}$ and $B : Y \longrightarrow \{0, 1\}$ be decision problems.

A is a polytime reducible to B, or "A reduces to B" ($A \propto B$), if there exists a function $R : X \longrightarrow Y$ that transforms inputs to A into inputs to B such that $A(x) = B(R(x))$.



Solving A is no harder than solving B. In other words, if solving B is "easy" (i.e. $B \in P$), then solving A is easy ($A \in P$). Equivalently, if A is "hard", then B is "hard". Given an algorithm for B, we can easily construct an algorithm for A.

Proving Celeste to be NP-Complete

The Game's Mechanics

The Game's Mechanics

Our inspiration - Erik Demaine

Our idea to classify Celeste was inspired from Erik Demaine's classification of Super Mario Bros.

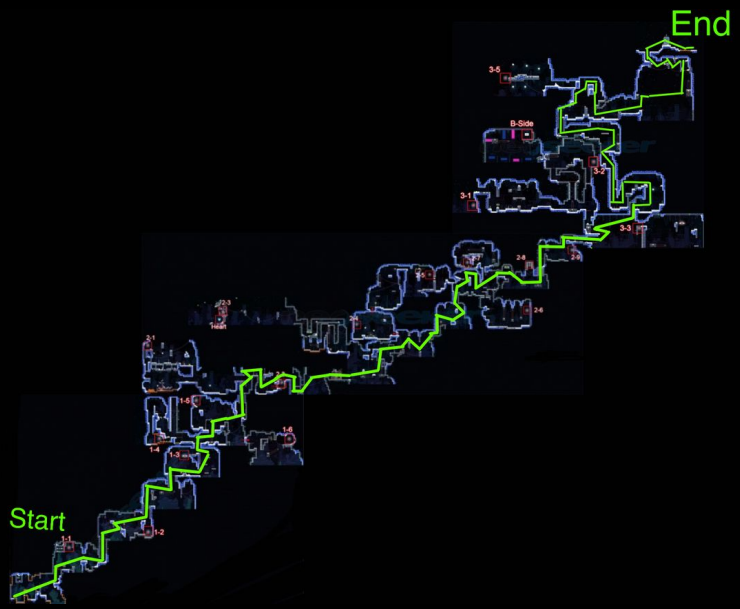
He proved a level of the game to be NP-Complete.

As a sequel to this, he also proved a generalised level of Super Mario Bros. to be PSPACE-Complete.

Celeste is NP

Given a level and a path, that is the moves required to reach the endpoint, You can verify if the path is correct just by applying those moves.

The moves are polynomial, why? We repeat the sections only after visiting other sections. Since the number of sections themselves are polynomial, we can only have polynomial moves before we complete the level. .

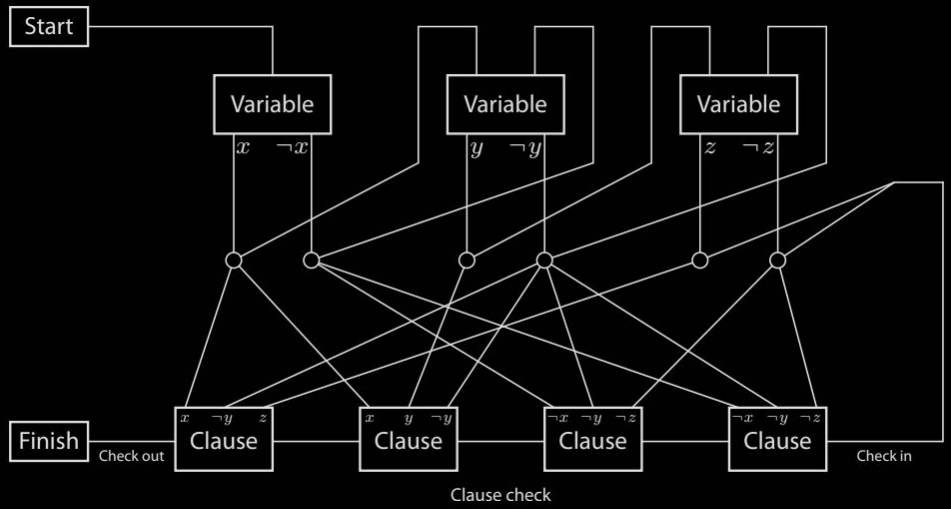


This clearly implies that the **game is NP**
For example, for the 1st level, we can map the path from start to end as seen below

Now since we have proven that Celeste is NP We can try to prove that it is NP-Hard, essentially proving that it is NP-Complete.

3-SAT Reductions

To prove the NP-Hardness of Celeste, we here describe a framework for reducing 3SAT to a 2-D platform game. This framework is based on https://arxiv.org/pdf/1203.1895.pdf. Using this framework in hand, we can prove the hardness of games by just constructing the necessary gadgets.



3-SAT Reductions

Required Gadgets:

Start and Finish: The start and end gadgets contain the spawn point and the end goal respectively.

Variable: Each variable gadget must force the player to make a binary choice (select x or ~x). Once a choice is taken the other choice should not be accessible. Each variable gadget should be accessible from and only from the previous variable gadget is such a way that it is independent of the choice of the previous gadget and going back is not allowed.

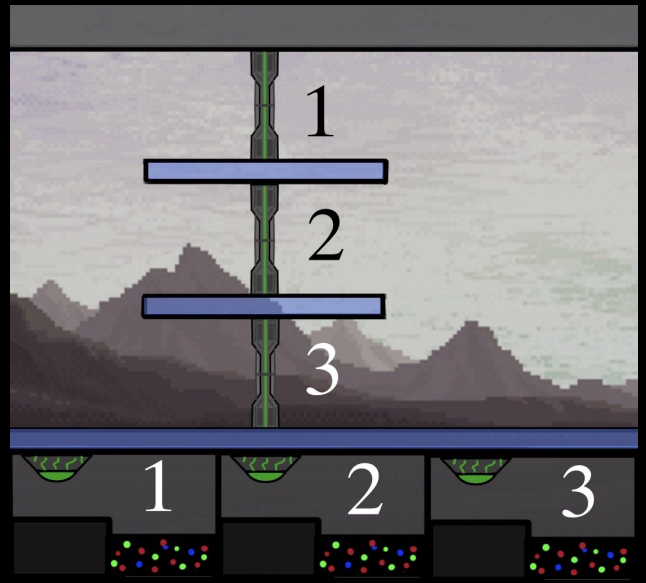
Clause: Each literal in the clause must be connected to the corresponding variable. Also, when the player visits the clause, there should be a way to unlock the corresponding clause.

Check: After all the variables are passed through, all the clauses are run through sequentially. If the clause is unlocked, then the player moves on to the next clause else loses.

Crossover: The crossover gadget allows passage via two pathways that cross each other. The passage must be such that there is no leakage among them.

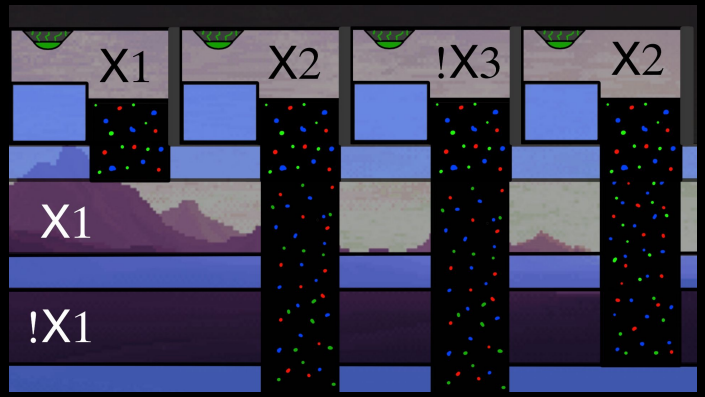
Variable and Clause Gadgets

Variable Gadget

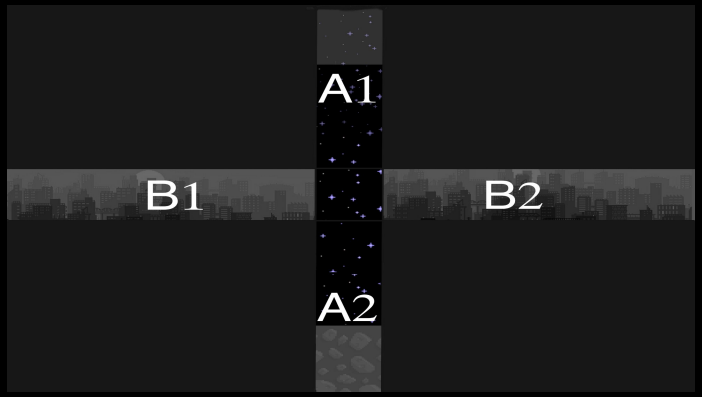


Clause Gadget

Tunnel and Crossover Gadget

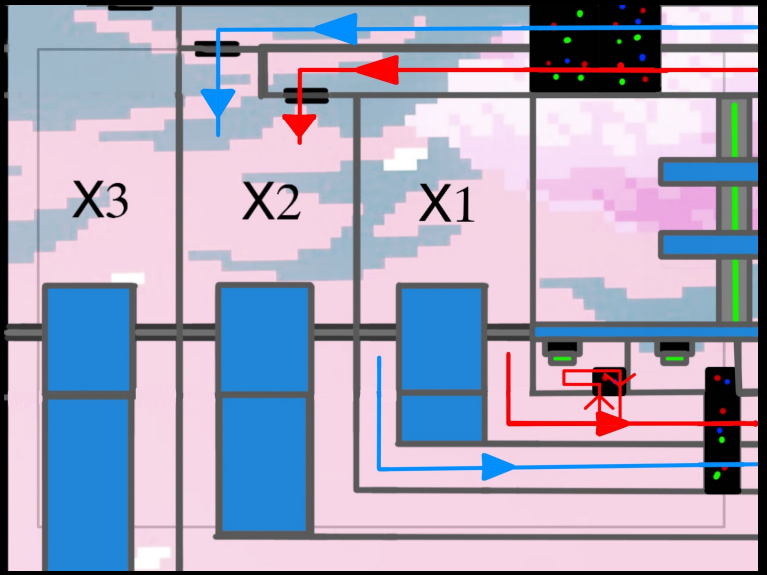


Tunnel Gadget

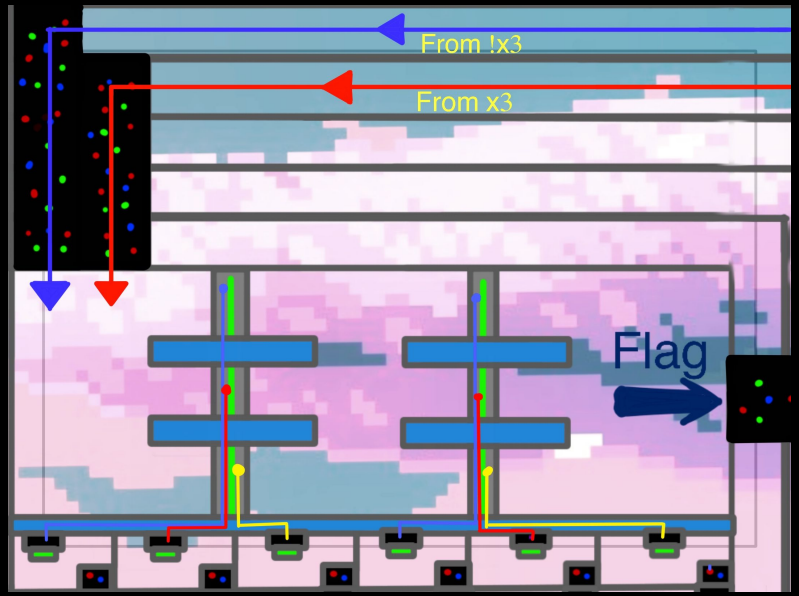


Crossover Gadget

Transitioning



Transitioning b/w Variable



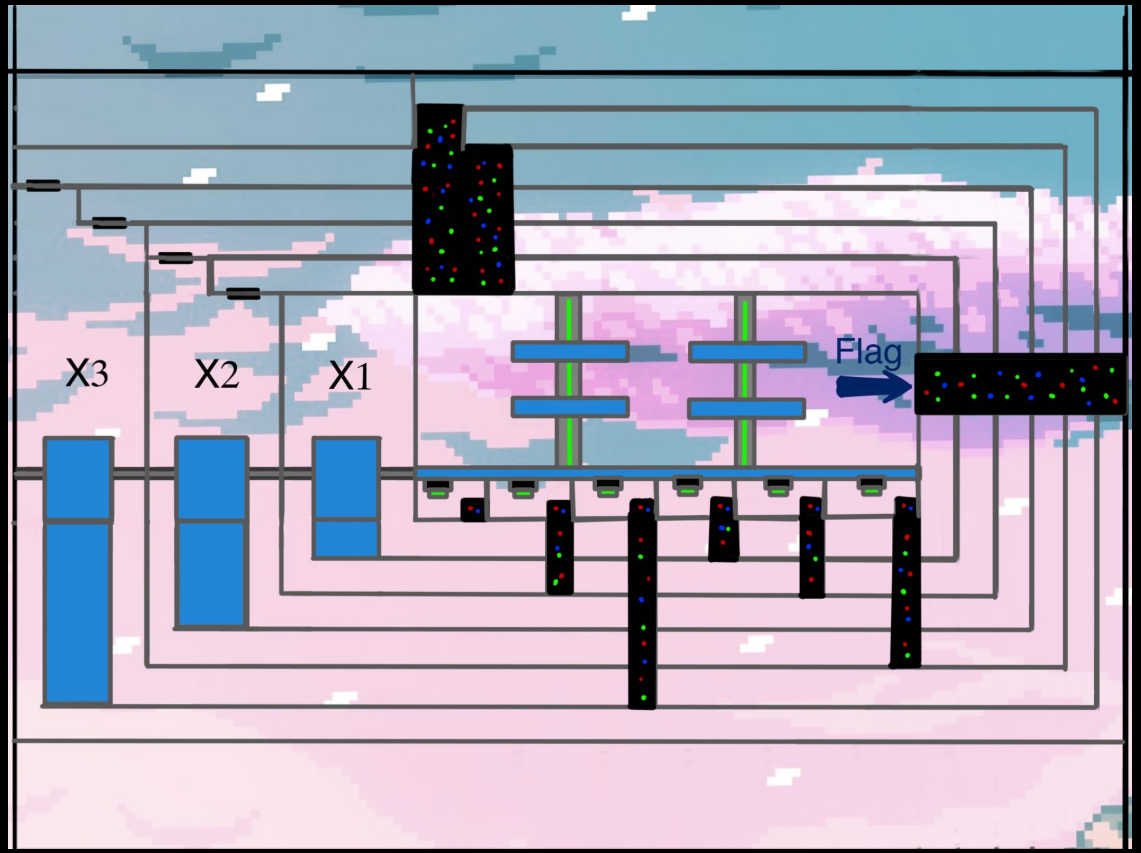
Final Passage

Our Level

We designed and reduced a level in Celeste to the 3CNF boolean expression:

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

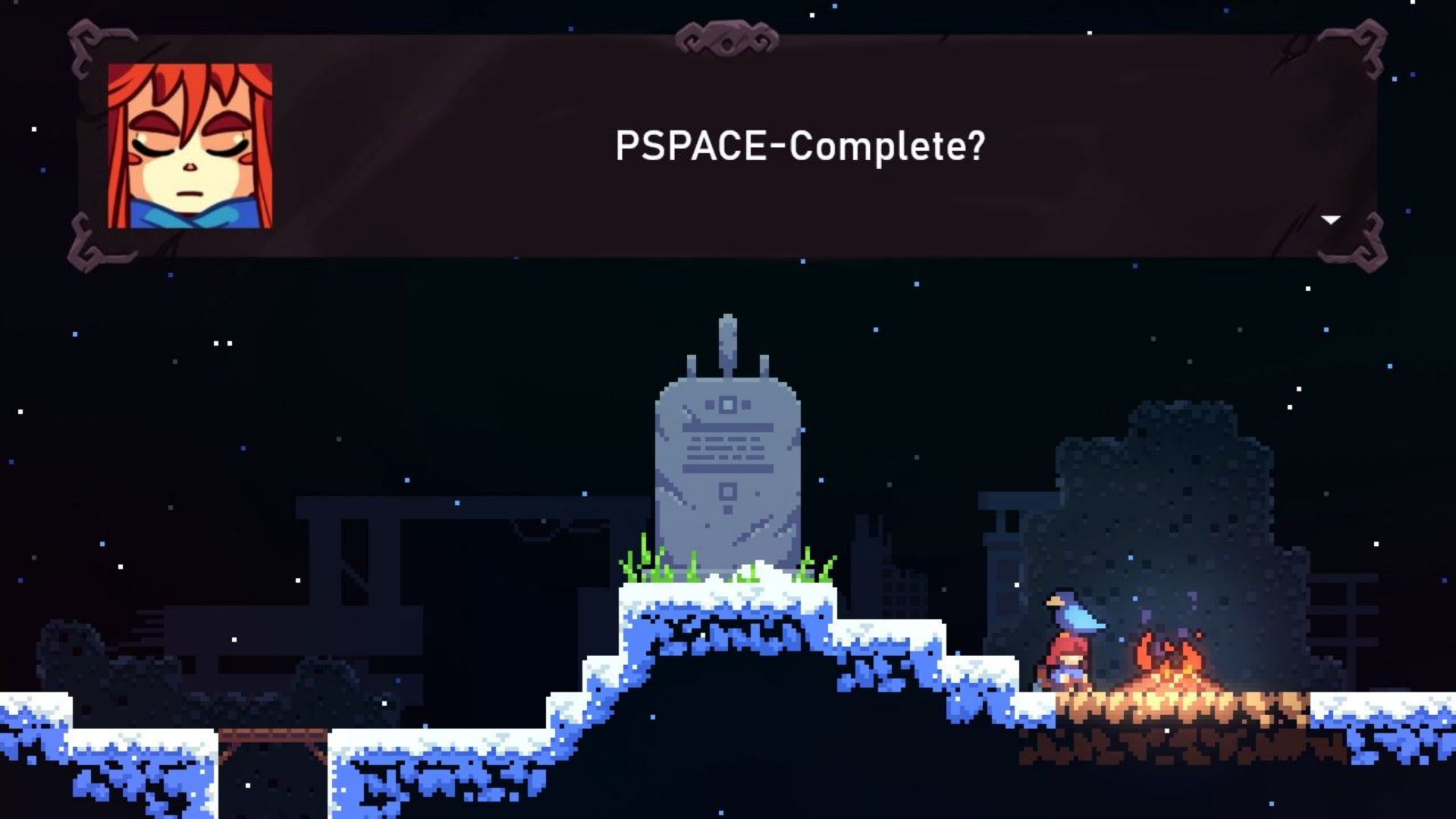
Hence, we proved that Celeste is at least as hard as 3-SAT.



Conclusion

- We proved that Celeste is at least as hard as 3SAT, making it NP-Hard.

- In conclusion, the game is both NP and NP-Hard, making it an NP-Complete puzzle.


PSPACE-Complete?

What is PSPACE-Complete?

If a decision problem in PSPACE and every other problem that can be solved PSPACE can be reduced to it in polynomial time , the problem is said to be PSPACE-complete.

This means that the PSPACE-complete are the hardest problems in PSPACE, as solution for this problem can be used to solve any other problem in PSPACE.

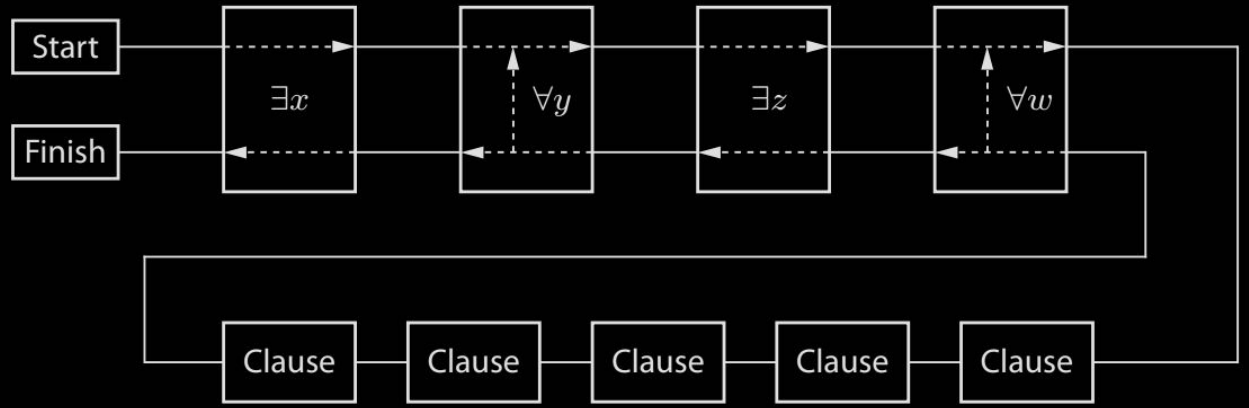
From Savitch's theorem, PSPACE is closed under nondeterminism, ie: PSPACE=NPSPACE. Hence PSPACE-complete are harder than NPSPACE problems too.

Additional features that Celeste needs

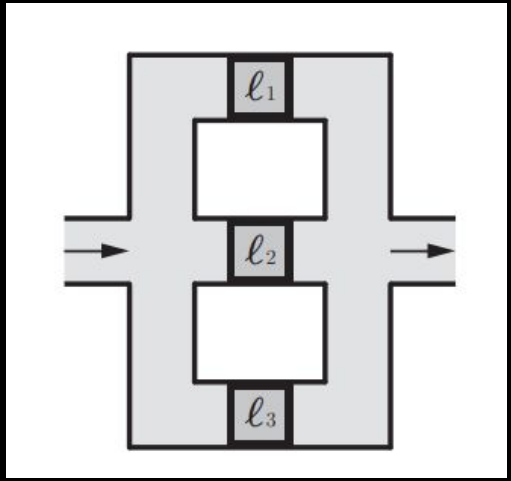
We now make an addition to the game. The door can now be closed using a red button. When the door is open, the red button is deflated and can be activated and when the door is closed, the green button can be activated.

Framework for PSPACE-Hardness

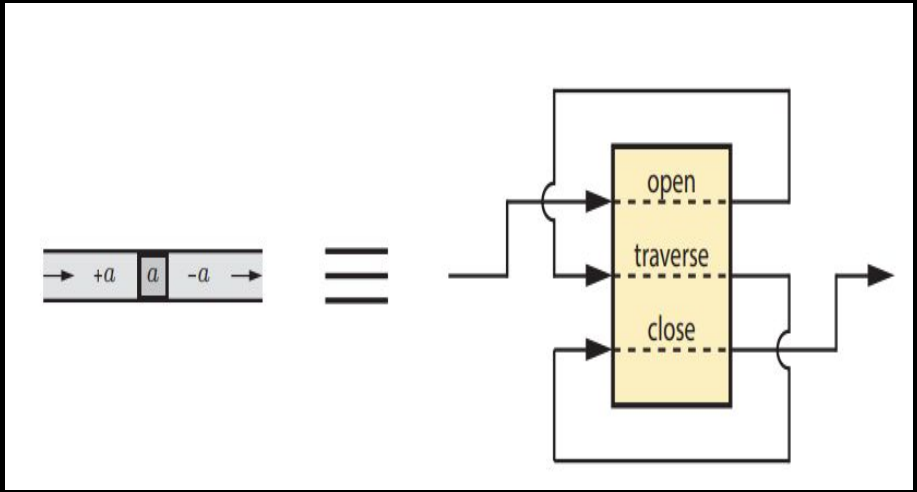
To prove the PSPACE-Hardness of Celeste, we here describe a framework for reducing TQBF to a 2-D platform game. This framework is based on the paper "Classic Nintendo Games are (Computationally) Hard". Using this framework in hand, we can prove the hardness of games by just constructing the necessary gadgets.



Gadgets for TBFQ Framework

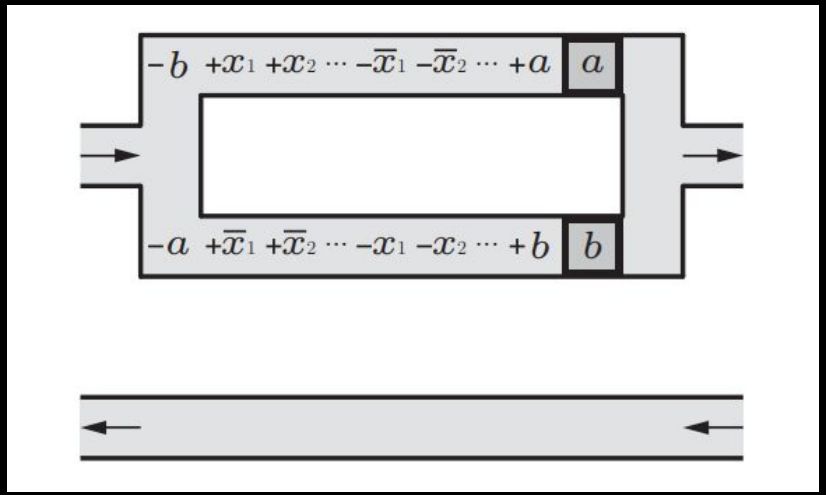


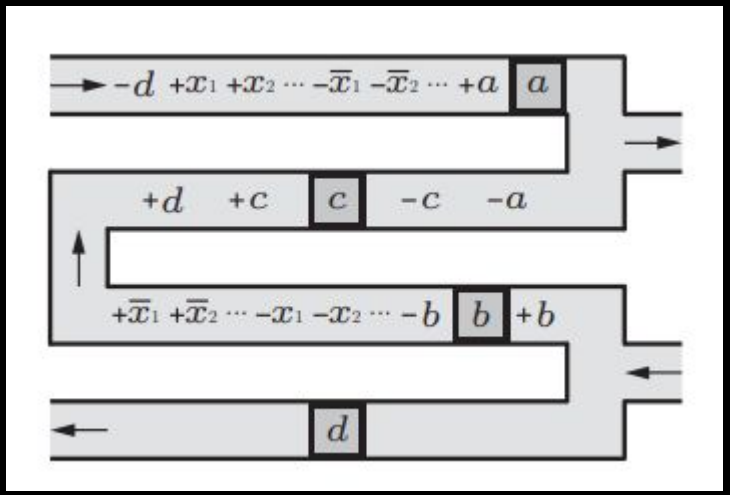
Clause Gadget



Shorthand notation for tunnel

Gadgets for TBFQ Framework





Framework Traversal

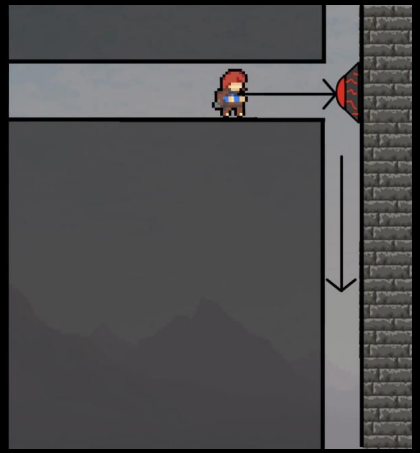
Traversing a quantifier gadget sets the corresponding variable in the clauses. When passing through an existential quantifier gadget, the player can set it according to their choice. For the universal quantifier gadget, the variable is first set to true.

A clause can only be traversed if at least one of the variables is set in it. After traversing all the quantifier gadgets, the player does a clause check and is only able to pass if all the clauses are satisfied. If the player succeeds, they are routed to lower parts of the quantifier gadgets, where they are rerouted to the last universal quantifier in the sequence.

The corresponding variable is then set to False and the clauses are traversed again. This process continues and the player keeps backtracking and trying out all possibilities.

Implementation of the Doors

For creating a door gadget we first need to create a way to force the player to dash into a button to activate it. We do this using a narrow tunnel and leaving only enough space to pass if the button is pressed.



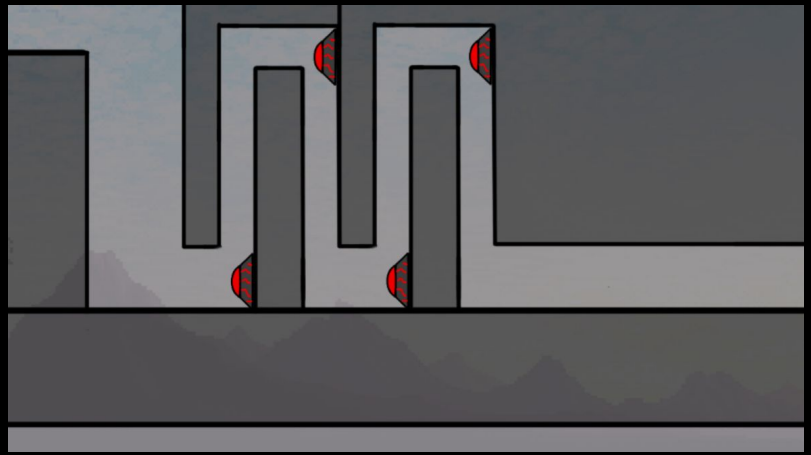
The button above will open the door, and the button below will close the door, and since the path is thin, Madeline will not be able to pass through until the button is pressed.

Tunnel Gadgets

To connect the Variable exit to the Buttons of the Clause, we use a Tunnel gadget.

Below the buttons, there are Tunnels leading from the exits of the variables. The variables have access to the buttons they can set true according to the boolean expression.

For example, if x_1 is chosen to be true, then Madeline gets access to the x_1 tunnel and $\neg x_1$ otherwise. x_1 tunnel has access to buttons that open a door to the clause having x_1.



But what exactly changed?

On adding the close button, we added a requirement to keep track of all the doors that are open. Once a door is opened, it always remained open. Before we had to only keep track of the current state sequentially as all the doors will be opened in a sequence. Knowing that a door is open implied that all the previous doors were opened so as to reach the current door.

But after adding the close button, at any point of the game, all the doors are independent and knowledge of the open state of a door gives us no info about the other doors. So, we have to keep track of all the other doors. This makes the game harder and makes its PSPACE-Hard instead of NP-Hard.

Lack of knowledge about the status of all the doors makes the game PSPACE-Complete

Conclusion

In conclusion, when we add additional feature, the game becomes both PSPACE and PSPACE-Hard, making it an PSPACE-Complete puzzle.

Game Theory

Game theory is a theoretical framework for conceiving social situations among competing players and producing optimal decision-making of independent and competing actors in a strategic setting.

Algorithmic game theory is a subset of Game Theory which deals with the design of algorithms in strategic environments with the help of game theory tools.

Computing properties of Nash Equilibria, Price of Anarchy and Best Response Dynamics are a few examples of famous problems that algorithmic game theory deals with.

Nash Equilibrium and Complexity Theory

Nash Equilibrium is a game theory concept that determines the optimal solution in a non-cooperative game in which each player lacks any incentive to change his/her initial strategy.

Approximate (mixed) Nash Equilibrium is shown to be PPAD Complete by Daskalakis, Goldberg and Papadimitriou.

We provide a proper intuition along with a suitable outline of this aforementioned proof among others (which include proving 2-NASH is PPAD).

Total Search Problems

EQUAL-SUBSETS and NASH form a very specific group of problems which appear to be different from typical NP-complete problems like SAT.

Why? Because unlike SAT, these problems have a guaranteed solution. These problems (EQUAL-SUBSETS, NASH etc.) thus are Total Search Problems - problems where every instance has a solution.

We have proved that Total Search Problems which are non-deterministic in nature are not NP-Complete.

Hard **TFNP** problems: an unhappy family

Happy families are all alike; every unhappy
family is unhappy in its own way.

— Leo Tolstoy

For our purposes:

NP-complete problems are all alike; every hard
TFNP problem is hard in its own way.

— don't quote me

PPAD class

PPAD stands for Polynoimal Parity Arguments on Directed Graphs. Complexity class.

PPAD is a subset of the class TFNP (Total Function in Non-deterministic Polynomial). PPAD is contained in PPA (Polynomial Parity Arguments on Undirected Graphs). However, it is not known if PPAD is equal to PPA.

Examples of PPAD-Complete problems include finding Nash Equilibria, fixed points in Brouwer functions. In a recent paper it was proved that Nash Equilibrium in a tree polymatrix game with 20 actions per player is PPAD-Hard.

Ok so...

MATH IS SLIDE-HARD

Nash Equilibrium

A Nash Equilibrium is a set of strategies - one strategy per player - such that no player has an incentive to unilaterally change his strategy (stable sub-optima). In case of two player zero-sum games, Nash Equilibrium is also optimal.

- **Pure** NE: where each player chooses a pure strategy
- **Mixed** NE: where for each player a probability distribution over his pure strategies is applied (in case of MNE we have expected payoff's associated with each player)

Formal Definition of Nash Equilibrium

From the above, $u_s^i = u_i(s)$ where $s \in S$.

An **individual mixed strategy** is a probability distribution on the set of available strategies. Such as in the last example, selecting one of "rock", "paper", or "scissors" uniformly at random is an example of a mixed strategy.

A **pure strategy** is simply a **special case** of a mixed strategy, in which one strategy is chosen 100% of the time.

- A **Nash equilibrium** is a strategy profile $s = (s_1, s_2, \ldots, s_n)$ with the property that,

$$u_i(s) \geq u_i((s_1, s_2, \ldots, s_i', \ldots, s_n))$$

  $\forall i$, where $s_i' \in S_i$ denotes a strategy other than s_i available to player i.

In the event that this inequality is strict, $u_i(s) > u_i((s_1, s_2, \ldots, s_i', \ldots, s_n))\ \forall i$, the profile s is called a **Strong Nash Equilibrium**. Otherwise, s is called a **Weak Nash equilibrium**.

Input: A game in normal form.

Output: Is there a Mixed Nash Equilibrium?

Now, in this problem for two-player games *the probabilities used by the players are rational numbers* (given that their utilities are also rational). So, it is clear how to write down the solution of a 2-player game.

However, as pointed out in Nash's original paper, when there are more than two players, there may be **only irrational solutions**.

As a result, the problem of computing a Nash equilibrium has to deal with issues concerning numerical accuracy. Therefore, we introduce the concept of Approximate Nash Equilibrium.

Approximate Nash Equilibrium

Input: A game in normal form where, $u_s^p \in [0, 1]$.

Output: Is there a Mixed ϵ-Nash Equilibrium?

$$\epsilon\text{-}\mathbf{Nash\ Eqm}\text{: } E[\text{payoff}] + \epsilon \geq E[\text{payoff}] \text{ of best possible response}$$

This should be at least as tractable as finding an exact equilibrium, hence any hardness result for approximate equilibria carries over to exact equilibria.

NASH: "Re"-Definition

> Bounded rationality fixes irrationality.

Given the *description of a game* (by explicitly giving the utility of each player corresponding to each strategy profile), and a *rational number* $\epsilon > 0$, compute a *(mixed) Approximate Nash Equilibrium*.

NASH is PPAD-complete

Here, we shall provide an outline to give an intuition behind why NASH is PPAD-complete.

Two parts of the proof are:

- Nash is in PPAD i.e., $\text{NASH} \preceq_p \text{END-OF-A-LINE}$
- $\text{END-OF-A-LINE} \preceq_p \text{NASH}$ or, NASH is PPAD-hard

Constraint Logic

We proved that Celeste, our original game of choice is NP-complete and in a different case PSPACE-complete by reducing it from Circuit Formalism of SAT and TQBF respectively. This is the more or less standard procedure in classic game complexity literature.

Constraint Logic, originally formulated by Erik Demaine, can be considered an intersection of Graph Theory and Circuit Formalism. We have what Demaine refers to as a constraint graph and rules that define legal moves on such graphs. The games serves as computation when it is played on a constraint graph and simultaneously "serves as a useful problem to reduce to other games to show their hardness".

Constraint Logic: Theory

A constraint graph serves as a model of computation, whose orientation describes the machine's state. What does that mean? Well, a machine in this computational model is an undirected graph with red and blue edges. A configuration of the machine gives directions to the edges, thus is a directed graph. Depending on the nature of the game (further constraints), the computation performed can be deterministic, non-deterministic, alternating etc. The constraint graph then accepts the computation just when the game can be won.

Definition: A *constraint graph* G is a directed graph, with edge weights $\in \{1, 2\}$. An edge is called *red* or *blue* respectively. The *inflow* at each vertex is the sum of the weights on inward-directed edges, and each vertex has a non-negative *minimum inflow* ($= 2$).

Legal Configuration: $\text{inflow}_i \geq \text{min-inflow}(= 2),\ \forall i \in V(G)$ which serves as the constraint. We can further assume, $\deg_i \leq 3,\ \forall i \in V(G)$ or in other words, G is 3-regular.

Legal Move: A legal move on a constraint graph is the reversal of a single edge, resulting in a legal configuration.

The Game: Reverse a given edge, by executing a sequence of moves. In multiplayer games, each edge is controlled by an individual player and each player has his own goal edge.

Deterministic Game: An unique sequence of reversals is forced.

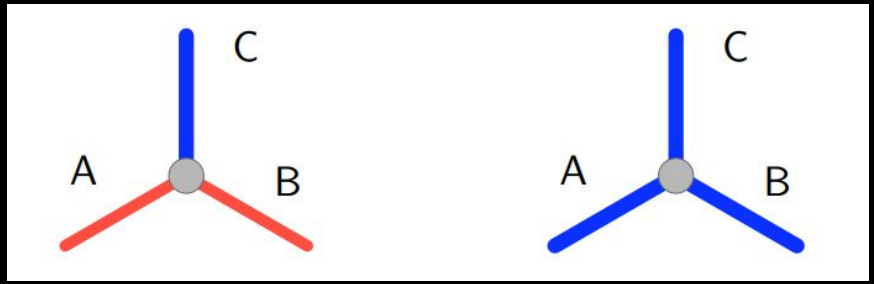
Bounded Game: Each edge may only reverse once.

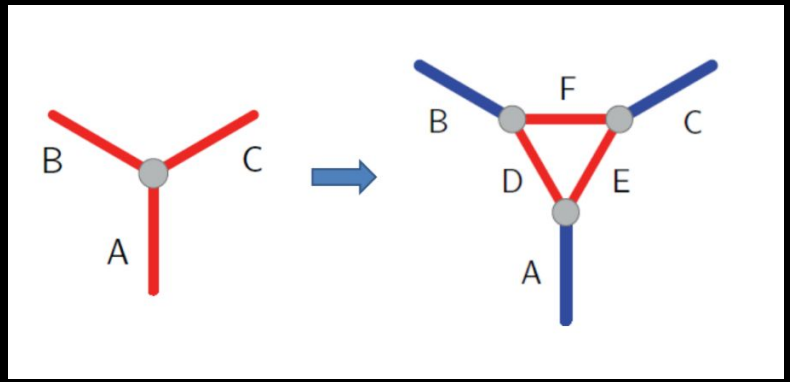
Major Theorems

1. **Existence**: The existence of a configuration for a given machine is NP-complete. A modified version would be to prove that Bounded Non-deterministic Constraint Logic is NP-complete.
2. **Reversibility Problem**: Given a constraint graph, can we reverse a specified edge by a sequence of valid moves, is PSPACE-complete.
3. **Reachability Problem**: The reachability problem, that is, given two configurations C_1 and C_2, can we reach C_2 from C_1 by a sequence of valid moves, is PSPACE-complete.

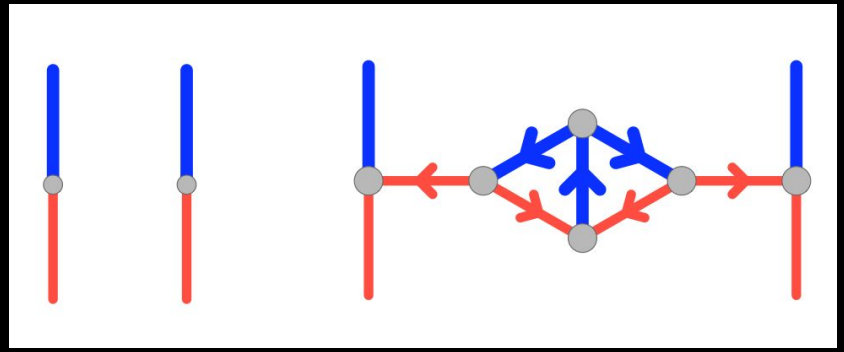
Note: The above two theorems hold for *planar graphs* in which each vertex touches either 3-*blue edges* or 2-*red*-1-*blue edges*.

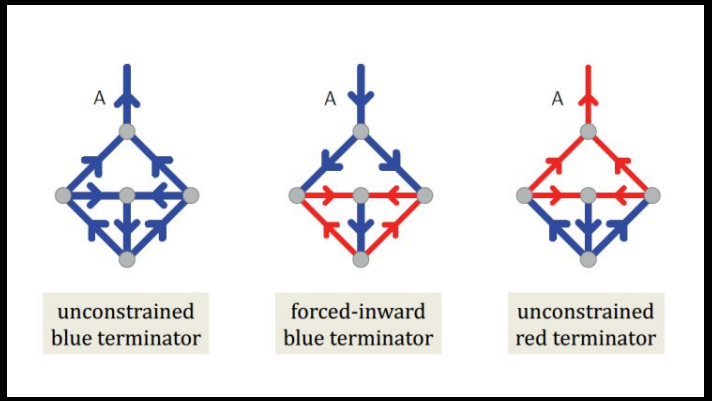
Constraint Logic



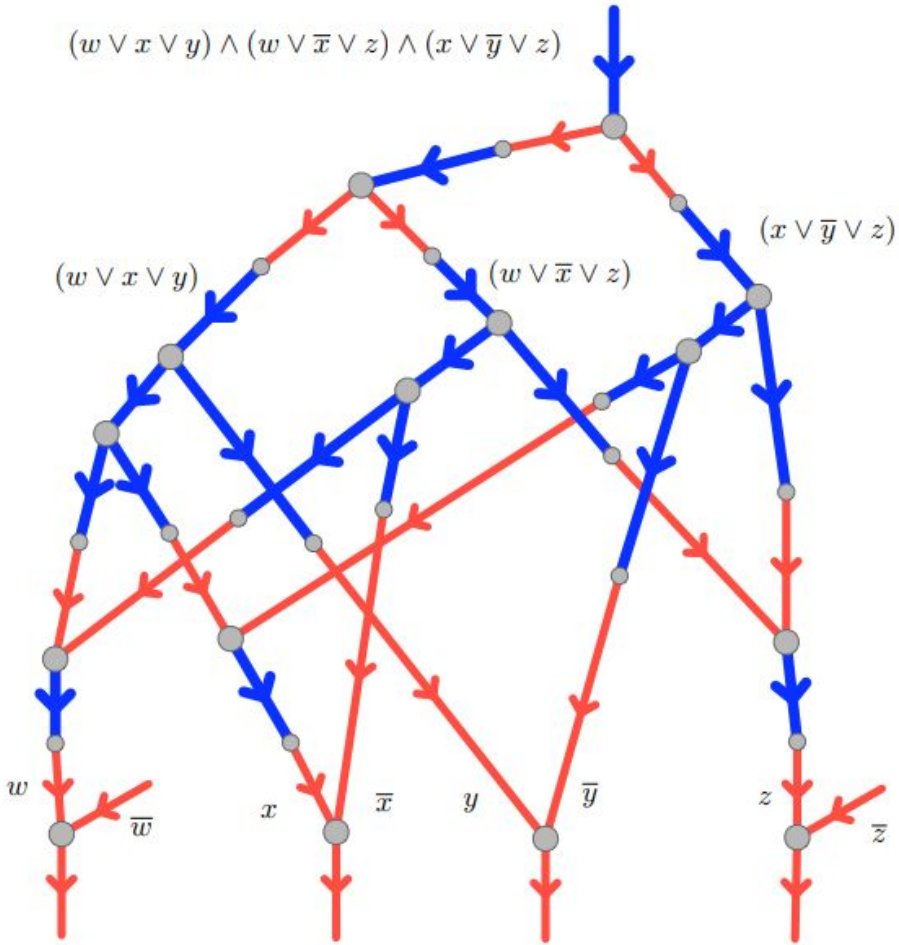


Constraint Logic

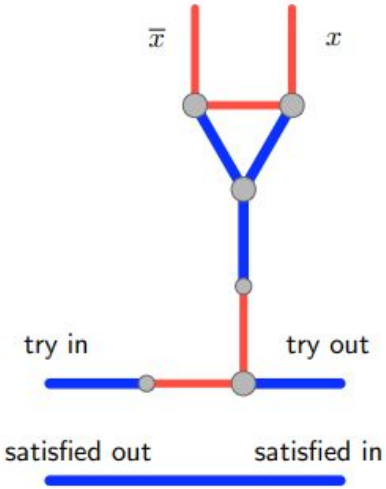


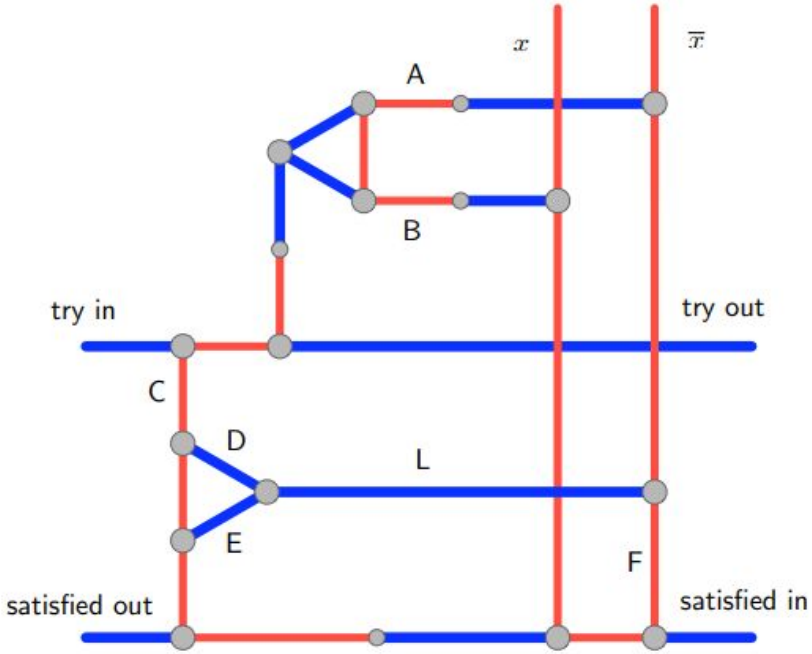


NP-complete b-NCL



PSPACE-complete NCL





2-Player Games

> I have come to the personal conclusion that while all artists are not chess players, all chess players are artists.
>
> **– Marcel Duchamp**

On that note, let's talk about chess. Suppose, that our realm in only 8×8 chess with all its classic rules, expect one - the 50 move rule, which we shall ignore here.

So, which class does 8×8 classic chess falls in? Well, it does fall in PSPACE as it takes constant space. But, that is not very interesting is it? So, what about $n \times n$ chess or generalised chess?

Theorems

1. *Bounded 2CL is PSPACE-complete.*
2. *2CL is EXPTIME-complete.*

Proof: Bounded NCL is NP-complete

Given an instance of 3-SAT, we construct graph G' as described above. Let F be the corresponding boolean formula. Then, clearly F is satisfiable, the CHOICE vertex edges may be reversed in correspondence with a satisfying assignment, such that the output edge may eventually be reversed. Similarly, if the output edge may be reversed, then a satisfying assignment may be read directly off the CHOICE vertex outputs. Now, $\exists$ an easy poly-time reduction from G' to G where we replace the vertices with 3-*blue-edge* vertex or 2-*red*-1-*blue-edges* vertex. Thus, Bounded NCL is NP-Hard.

Thus, a proper interpretation of the above proof should be that **Bounded NCL is NP-complete**. Moreover Bounded NCL is NP-complete for planar graphs as well.

This proof serves as the backbone for proving that Bounded Puzzles are NP-Complete (for example, the first version of our CELESTE proof).

Proof: NCL is PSPACE-complete

NCL is in PSPACE because the state of the constraint graph can be described in a linear number of bits, specifying the direction of each edge, and the list of possible moves from any state can be computed in polynomial time.
Thus, we can have an NPSPACE algorithm where we non-deterministically traverse the state space while maintaining only the current state. Now, by Savitch's Theorem ($\mathrm{PSPACE} = \mathrm{NPSPACE}$) for this NPSPCE algorithm there exists a PSPACE algorithm into which the previous can be converted.

Moreover, NCL is also PSPACE-hard because deciding whether an edge may reverse also decides the TQBF problem. For the complete proof we require gadgets for the quantifiers.

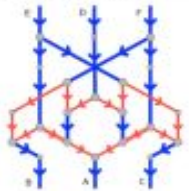
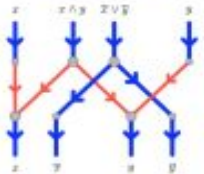
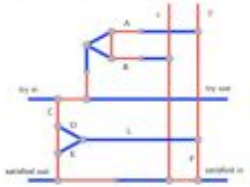
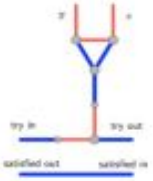
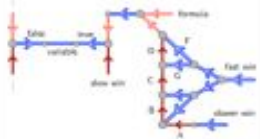
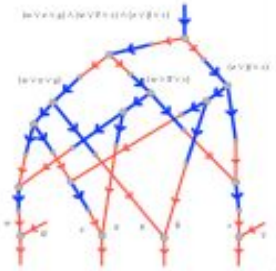
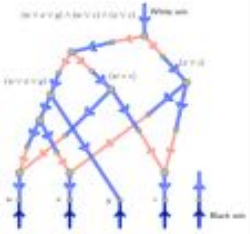
Simulations

One of the most famous simulations is Conway's Game of Life. This game in our formulism is a Unbounded DCL or Deterministic Constraint Graph. Just as Non-deterministic Constraint Logic serves as the one-player version of Constraint Logic, the zero-player version is DCL (as expected).

Let's talk about the results for simulation:

1. *Bounded DCL is P-complete*: This result is quite boring for the simple fact that there are no interesting games that tend to lie in this genre.
2. *DCL is PSPACE-complete*: This result can be used to model Game of Life and provide a simpler proof as to why it is PSPACE-complete.

Constraint Logic

	Zero player (Simulation)	One player (Puzzle)	Two player	Team, Imperfect information
Unbounded	PSPACE  	PSPACE  	EXPTIME 	Undecidable Undecidable
Bounded	P **404**	NP 	PSPACE 	NEXPTIME NEXPTIME

Design Analysis of 2D-platform games

We have analysed the design aspects of 2D-platform games as they are very unique in the sense that different components of the game can be used to form complex puzzles. They are often complete for either NP or PSPACE.

This completeness analysis is often found un-applicable in modern games, especially non-platform games.

Generalizations in 2D platform games

Video games are decision problems, with fixed states. Generalisation of the features common to many games helps in understanding and approaching the problem with greater abstraction.

While certain or combination of features are easy from an algorithmic point, other features, or certain combinations of thereof, are hard to manipulate.

Having a hard feature within the game essentially imply that the game itself is hard to solve, regardless of other details.

This abstraction can be used to form "meta-theorems" which hold for all games having the features. For any given game, these meta-theorems can be adjusted according to details of the particular game.

Metatheorems

1. *A 2-D platform game where the levels are constant and there is **no time limit** is in **P**, even if the collecting items feature is present.*
2. *Any game exhibiting both **location traversal** (with or without a starting location or an exit location) and **single-use paths** is **NP-hard**.*
3. *Any 2-D platform game that exhibits the features l**ong fall and opening doors** is **NP-hard**.*
4. *Any 2-D platform game that exhibits the features **long fall, opening doors and closing doors** is **PSPACE-hard**.*
5. *A game is **NP-hard** if either of the following holds:*

   *(a) The game features **collectible** tokens, toll roads, and location traversal.*
   *(b) The game features **cumulative** tokens, toll roads, and location traversal.*
   *(c) The game features **collectible cumulative** tokens, toll roads, and the avatar has to reach an exit location.*
6. *A game is **NP-hard** if it contains **doors and one-way paths**, and either of the following holds:*

   *(a) The game features **collectible** keys and location traversal.*
   *(b) The game features **cumulative** keys and location traversal.*
   *(c) The game features **collectible cumulative** keys and the avatar has to reach an exit location.*

7. *If a game features **doors and pressure plates**, and the avatar has to reach an exit location in order to win, then:*

   *(a) Even if **no door** can be **closed by a pressure plate**, and if crossovers are allowed, then the game is **P-hard**.*

   *(b) Even if **no two pressure plates** control the same door, the game is **NP-hard**.*

   *(c) If **each** door may be controlled by **two pressure plates**, then the game is **PSPACE-hard**.*

8. *If a game features **doors and k-buttons**, and the avatar has to reach an exit location in order to win, then:*

   *(a) If **k > 1,** and crossovers are allowed, then the game is **P-hard.***
   *(b) If **k > 2,** then the game is **NP-hard.***
   *(c) If **k > 3,** then the game is **PSPACE-hard.***

9. *There exists an NP-complete game featuring doors and 2-buttons in which the avatar has to reach an exit location. (One that we end up proving in CELESTE).*

Conclusions

Games serve as models of computation which have been used quite prevalently used to mathematically model real-life scenarios. For example, classical game theory deals with several games involving rational decision making strategies and has found applications in computer science, biology and social sciences.

Games not only serve as a means of understanding computation but also decision making and behavioral relations. This is why they have been often found associated with numerous breakthroughs in artificial intelligence. It is quite astonishing as how games often are found to be embedded in deep computational problems and yet require incredibly less to none formal understanding to be played.

Conclusions

Our complexity journey was a beautiful one and we plan to pursue it in the future too.

DEDICATED TO

Alan Turing

Erik Demaine

"And everyone else for bearing with me" (- Alapan)

Thank You

Alapan Ashwin Hrishi Kunwar Kushagra
Manasvi Pulak Shreyas Zeeshan