Bounded Rationality of Restricted Turing Machines*

Lijie Chen Pingzhong Tang Ruosong Wang
Institute for Interdisciplinary Information Sciences,
Tsinghua University, Beijing, China.
{chenlj13,wrs13}@mails.tsinghua.edu.cn, kenshinping@gmail.com

Abstract

Bounded rationality aims to understand the effects of how limited rationality affects decision-making. The traditional models in game theory and multiagent system research, such as finite automata or unrestricted Turing machine, fall short of capturing how intelligent agents make decision in realistic applications. To address this problem, we model bounded rational agents as restricted Turing machines: restrictions on running time and on storage space. We study our model under the context of two-person repeated games. In the case where the running time of Turing machines is restricted, we show that computing the best response of a given strategy is much harder than the strategy itself. In the case where the storage space of the Turing machines is restricted, we show the best response of a space restricted strategy can not be implemented by machines within the same size (up to a constant factor). Finally, we study how these restrictions affect the set of Nash equilibria in infinitely repeated games. We show restricting the agent's computational resources will give rise to new Nash equilibria.

1 Introduction

Bounded rationality has been a topic of extensive interest in artificial intelligence and multi-agent system research [11, 12, 3, 23, 24, 4, 5, 22, 21]. It refers to the limitations (time, space, information, etc) agents encounter that prevent them from making a fully rational decision in realistic settings. This phenomenon has been widely studied in the realm of repeated games [16]. An important feature of repeated games, often modeled as extensive-form games, is their gigantic strategy space. A strategy of a player needs to specify his action choice for any possible history (on or off the actual play path) where it is his turn to move. This leads to the difficulty that the description of a general strategy costs exponential bits in storage and is highly unrealistic. To mitigate this difficulty, a stylized approach models such strategies as *finite automata* [18, 16], where "equivalent" histories are grouped into state. Under this compact formulation, the set of equilibria has been characterized [18], the computation of the best response against an automata strategy has been investigated [7, 2] and the computation of the Stackelberg equilibrium has been investigated [25].

However, restricting strategies to finite automata loses generality. For example, in infinitely repeated prisoner's dilemma, to model the following strategy:

`Play D iff the other played D more than C in the past`,

*This work was supported by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the Natural Science Foundation of China Grant 61033001, 61361136003, 61303077, 61561146398, a Tsinghua Initiative Scientific Research Grant and a China Youth 1000-talent program.

one must resort to machines such as pushdown automata.

Indeed, in reality, one can do much better than finite automata: we write computer programs! This inspires researchers to consider the possibility of modeling the bounded rational agents as general Turing machines. Megiddo and Wigderson [13] model a strategy as a general Turing machine and show that, in finitely repeated games, computing best response against such a machine is trivial by another Turing machine. Knoblauch [9] shows that in infinitely repeated games and limit-of-means utility, there exists a Turing machine strategy such that no Turing machine can implement its best response. Nachbar and Zame [15] derive the same results, for discounted utility.

However, the general Turing machine model is also unrealistic in that it assumes an agent can perform computation in arbitrarily long time and use arbitrarily large storage space. Taking this into consideration, existing work has investigated Turing machines with size-restrictions (aka. restrictions on Kolmogorov complexity). Megiddo and Wigderson [13] show that, under a certain hypothesis, cooperation can be approximately sustained in repeated prisoner's dilemma if we restrict the size of a Turing machine. Lacote [10] later shows that cooperation can be sustained in finitely repeated games if and only if the Kolmogorv complexity of one player's strategy is substantially smaller than the number of rounds.

In this paper, we explore this direction further, by studying a realistic model of bounded rationality, where agents are confined to use time-restricted or space-restricted Turing machines to implement their strategies. We first use computational complexity models to rigorously define time and space restrictions. We then study the important game theoretical question of how to compute and implement the best response of such a restricted Turing machine.

For time-restricted case. We show that computing best response against a strategy whose running time is bounded by a polynomial in the number of rounds is NP-*complete*, more generally, computing best response against a strategy with oracle access to a $\sum_{\mathsf{i}}^{\mathsf{P}}$-*complete* language whose running time is bounded by a polynomial in the number of rounds is $\sum_{\mathsf{i+1}}^{\mathsf{P}}$-*complete*. Readers may refer to Appendix A or a standard computational complexity textbook (e.g., [1]) for the definition of complexity class $\sum_i^{\mathsf{P}}$.

The above results suggest that finding the best response of a strategy is harder than the strategy itself. It also suggests even if your opponent runs some polynomial time algorithm to decide its decision, you might not be able to efficiently find the best response against him under the conjecture that $\mathsf{P} \neq \mathsf{NP}$.

We study the space-restricted case under two natural models and show that, computing its best response is PSPACE-*complete*. We also show that under one of these models, implementing a strategy's best response requires a super linear expansion in strategy size under a certain reasonable complexity conjecture.

Those results suggest that, in contrast to time-restricted case, finding the best response in polynomial space is possible, but implementing them with linear expansion in size is impossible.

The second question we study is that, if both players have bounded rationality and this is common knowledge, how does it affect on the play of the game? More specifically, how does it change the set of the Nash Equilibria? We show that, in infinitely repeated games, interesting new equilibria will emerge. The intuition behind this result is as follows: For certain strategies, when assuming unbounded computational power of the opponent, these strategies will yield low utilities; however, knowing (by the common knowledge assumption) that the opponent is restricted, these strategies guarantee high utilities and can emerge as new Nash equilibrium! The proof of this part is quite nontrivial and is of independent interest.

1,1	0,5
5,0	3,3

Table 1: Payoff matrix of Prisoner's Dilemma

2 Preliminaries

2.1 Repeated Games

In this paper, we focus on two-person repeated games.

Definition 1. *$G = \langle S_1, S_2, u_1, u_2 \rangle$ is a two-person game in normal form. S_i is the finite set of actions for player i and $u_i : S_1 \times S_2 \to \mathbb{R}$ is the utility function for player i.*

Definition 2. *A super game G^n consists of n consecutive repetitions of a stage game G. At the t-th repetition, each player's strategy is to choose an action based on the history plays in rounds $1 \dots t-1$. That is, a player's strategy in the super game is a function that maps the set of all possible histories to the set of actions.*

Definition 3. *In a super game G^n, denote s_i as the strategy of player i. The utility for player i is $U_i(s_1, s_2) = \sum_{t=1}^{n} u_i(a_{t,1}, a_{t,2})$, where $a_{t,i}$ is the action of the player i at round t.*

For ease of exposition, we consider the simplest non-trivial case where the stage game is the well-known Prisoner's Dilemma whose payoff matrix is given in Table 1. Sometimes we may call Prisoner's Dilemma PD game for short. In the remainder of this paper, we use G to denote the Prisoner's Dilemma. We call the two actions of a player *cooperate* and *defect*. Map cooperate to 1 and defect to 0, a strategy is then equivalent to a function: $\{0,1\}^* \to \{0,1\}$.

For brevity, we denote *Turing machine, deterministic Turing machine and nondeterministic Turing machine* by TM, DTM and NTM, respectively. Readers can refer to Appendix A for a list of self-contained conceptions and definitions related to computational complexity, e.g., the definition of time constructible function, the time/space restricted complexity class DTIME, NTIME, DSPACE, NSPACE, PSPACE and $\mathsf{LOGSPACE}$, the polynomial hierarchy (PH) with related complete problem $\sum_i^{\mathsf{P}} \mathsf{SAT}$ and the definition of oracle machine $\mathsf{P}^{\mathcal{O}}$.

Definition 4. *For a strategy s, define the language of s to be the set of histories based on which s plays cooperate. Here the history of a repeated game before the t-th round is the string $a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}, \dots, a_{t-1,1}, a_{t-1,2}$ in which $a_{i,j}$ is the action that the player j takes in round i. We say a TM M implements a strategy s if M decides the language of s.*

We define a strategy's complexity by its language's complexity class.

Definition 5. *Let $\mathcal{C}$ be a complexity class. A strategy s is a $\mathcal{C}$-strategy if the language of s belongs to $\mathcal{C}$.*

Following the definition above, it is natural to further define time-restricted strategies like P-strategy and space-restricted strategies like PSPACE-strategy, which will be studied in the following sections. Also, we say a strategy s is a TM-strategy if the language of s is a computable language.

3 Time-restricted Strategies

In this section, we study how much time resource is needed to find or implement a best response of a time-restricted strategy.

As we do not care about the amount of space resources used, implementing a best response of a particular strategy in a super game is simple as one can simply store the optimal action sequence and play according to that. Thus, we focus on studying the computational complexity of finding the best response in a super game G^n against a time-restricted strategy.

Our plan of computing the best response of a polynomial-time strategy (i.e., P-strategy) consists of two steps. First, we compute the cumulative utility of the best response. Second, we construct the best response based on the utility computed during the first step.

The decision version of the first step is whether there exists a strategy that can gain at least utility k from the strategy represented by a polynomial-time TM M? However, this question is in fact not well defined. M may run in super-polynomial time, or even never halts. Meanwhile, by Rice theorem [17], deciding whether a TM M runs in polynomial time or always halts on all inputs is uncomputable. Thus, to make this problem computable, we have to put some restriction onto it.

To address the issue mentioned, we resort to complexity theory and restrict the TM's running time explicitly by a time constructible function f.

Definition 6. *Let f be a time constructible function and M be a TM, define M_f as a TM such that it runs M on input string x of length n for $f(n)$ steps. During the $f(n)$ steps, if M halts, then M_f return the output of M; otherwise M_f rejects.*

It is easy to see that M_f represents a strategy since it always halt. In addition, if f is a polynomial, M_f is indeed a P-strategy.

Let f be a time constructible function, define the decision problem BR_f as follows.

Definition 7. $\mathsf{BR}_f = \{\langle M, 1^n, k\rangle\}$ *such that there exists a strategy that can gain at least utility k against the strategy M_f in the game G^n.*

Here we write n in unary form as if we write n in binary form instead, the best response sequence will have exponential length with respect to the input size.

We first show how to use an oracle to the decision problem to find the best response.

Lemma 1. *For a given strategy M whose running time is bounded by f, finding the best response sequence is in $\mathsf{P}^{\mathsf{BR}_f}$.*

Proof. Clearly, the maximum utility we can achieve is $5n$. We enumerate the maximum utility and check them one by one using the oracle to BR_f. After we get the maximum utility, we then generate the best response sequence using the algorithm described below.

Start by the first round, we first emulate M to get the opponent's action of the first round. Denote the opponent's action by a. Then we try both actions. For each action a', we construct another Turing machine M' such that M' uses (a, a') as the history of the first round and then invokes M. M' returns the output of M. Now we have a $n-1$ round super game and we call the oracle to BR_f to figure out which action leads to the maximum utility. Fixing this action as the best response for round 1 and continue this procedure for the following rounds we can then get the best response sequence in $\mathsf{P}^{\mathsf{BR}_f}$. □

With Lemma 1, it suffices to study the complexity of the decision problem BR_f. We have the following theorem.

Theorem 1. *There is a polynomial f such that BR_f is NP-complete. For every polynomial f, BR_f is in NP.*

Our plan is to reduce SAT to BR_f. Given a CNF formula φ, the intuition here is to construct an agent that cooperates only if the opponent finds a satisfying assignment for φ. It treats the opponent's actions in the first n rounds as the assignments to the n variables of the SAT instance and checks its validity. If the opponent gives a satisfying assignment for φ, then it cooperates in the following rounds. Otherwise, it defects and thus induces a low cumulative utility for the best response. The detailed proof comes as follow.

Proof. Clearly BR_f is in NP for every polynomial f. A polynomial witness is the action sequence yielding utility k.

To show NP-hardness, we reduce a SAT problem instance φ to a BR_f problem instance.

Given a CNF formula φ with n variables and the binary description length of φ is $|\varphi|$. Note that $|\varphi| \geq n$ Consider the following TM M which takes φ and the set of histories u as input:

- If $|u| < 2|\varphi|$, then M cooperates.
- If $|u| \geq 2|\varphi|$, then M extracts the opponent's actions in the first n rounds of the history u as $x_1, x_2, \cdots, x_n$, and checks whether x is a valid assignment for φ, if it is a valid assignment, then M cooperates (by accepting), and defects (by rejecting) otherwise.

Clearly M runs in polynomial time. Let us say its running time is bounded by polynomial $g(n)$.

Fixing the CNF formula φ for M we can then construct another TM M_φ which always uses φ as the input of M. Thus, the only input of M_φ is now the history u. To analyze the time complexity of M, we consider the following two cases.

- Case 1. When the input length $|u| < 2|\varphi|$, M_φ halts when finishing counting the length of the input, thus runs in $f_1(|u|)$ steps where $f_1(|u|)$ is a polynomial which doesn't depend on $|\varphi|$.
- Case 2. When the input length $|u| \geq 2|\varphi|$, we add the description of φ to the front of the input and invoke M. The running time can be bounded by $f_2(|u|)$ where $f_2(|u|)$ is a polynomial which does not depend on $|\varphi|$.

To summarize, we conclude that for any φ, M_φ runs in time bounded by a fixed polynomial $f = f_1 + f_2$ which doesn't depend on φ. This f is the polynomial we specify for BR_f.

Now considering $x = \langle M_\varphi, 1^{10m}, 27m \rangle$, if φ has a solution, then the best response will achieve utility $27m$ in the last $9m$ rounds, so $x \in \mathsf{BR}_f$. Otherwise, we can at most get $5m + 9m = 14m$, so $x \notin \mathsf{BR}_f$.

It is clear that this reduction can be done in polynomial time, which concludes the proof. □

A direct corollary of Theorem 1 is that for any polynomial g that is always greater than f, BR_g is NP-*complete* as well.

As for a P-strategy, computing the utility of its best response is in NP, then by Lemma 1, we know that we can find the strategy itself in P^{NP}. Meanwhile, Theorem 1 suggests that, in order to compute the best response for a general P-strategy, one must be within the class of P^{NP}.

The natural next question to ask what is the complexity of finding the best response against a P^{NP}-strategy. With the notation of oracle machine, we can generalize Definition 6 and Definition 7 as follows.

Definition 8. *Let f be a time constructible function and $\mathcal{O}$ be a language, $M^{\mathcal{O}}$ is a TM with oracle access to $\mathcal{O}$. Define $M_f^{\mathcal{O}}$ as a TM with oracle access to $\mathcal{O}$ such that it runs $M^{\mathcal{O}}$ on input x of size n for $f(n)$ steps. During the $f(n)$ steps, if $M^{\mathcal{O}}$ halts, then $M_f^{\mathcal{O}}$ return the output of $M^{\mathcal{O}}$; otherwise $M_f^{\mathcal{O}}$ rejects.*

Definition 9. $\mathsf{BR}_{f,\mathcal{O}} = \{\langle M, 1^n, k\rangle\}$ *such that there exists a strategy can yield at least utility* k *against the strategy* $M_f^{\mathcal{O}}$ *in the game* G^n.

Using a similar reduction method as have used in the proof of Theorem 1, we can further prove the following corollary.

Corollary 1. *There is a polynomial* f *such that* $\mathsf{BR}_{f,\sum_i \mathsf{SAT}}$ *is* $\sum_{\mathsf{i+1}}^{\mathsf{P}}$*-complete. For all polynomials* f, $\mathsf{BR}_{f,\sum_i \mathsf{SAT}}$ *is in* $\sum_{\mathsf{i+1}}^{\mathsf{P}}$.

Combining Corollary 1 and the definition of PH (Definition 33), we can deduce that finding the best response for PH-strategy is also in PH. Furthermore, if PH does not collapse, then find the best response for a PH-strategy is harder than the strategy *per se*.

4 Space-restricted Strategies

In this section, we study the space restricted strategies. The first natural idea is to study the space complexity to calculate the best response against a PSPACE-strategy.

Lemma 2. *The best response of a* PSPACE*-strategy can be found in* PSPACE.

Proof. We enumerate all our possible action sequences and pick the best one. This algorithms needs only n additional space to store the current action sequence, and thus is in PSPACE. □

However, PSPACE-strategy is unrealistic in practice due to the huge amount of space owned by the player. Thus, we switch to study LOGSPACE-strategies that players are limited to use logarithm amount of extra space to calculate the best response.

For LOGSPACE-strategies, notice that the history has large length, which makes it possible for a strategy to "cheat" by gaining extra space via outputting extra information into the history. The idea is to transform a polynomial time TM to a polynomial size circuit in LOGSPACE. At every step, we evaluate one gate in this circuit and output as an action of the LOGSPACE-strategy. Now the game has polynomial number of rounds. By such a method, a LOGSPACE-strategy can do something similar to a P-strategy.

The above result for LOGSPACE-strategies is not interesting. The reason is that, since the space is limited, it is dubious that the strategies afford to store the whole history. Thus, We need to find better models to study space-restricted strategies, in which the whole history is not revealed to the strategy directly.

We propose the following alternative models. The main idea is to model space-restricted strategy as a function that takes the last action of the opponent and the current information bits as input, and outputs the new information bits and the action of this round.

An N-bits-strategy is a function $\mathit{trans} : \{0,1\}^{N+1} \to \{0,1\}^{N+1}$. The function *trans* takes the information bits and the action of the opponent in last round as input, outputs the information bits and the action for the current round.

The game is played as follows. See Figure 1 for an illustration. Let o_i be the action of the opponent in the i-th round and a_i be the action to be played in the i-th round. We assume $o_0 = 0$. Let b_i be the information bits outputted after the i-th round. We assume $b_0 = 0^N$. In the i-th round, we calculate use the *trans* function to calculate the new information bits and the action, i.e., $(a_i, b_i) = \mathit{trans}(o_{i-1}, b_{i-1})$.

We remark that this strategy in fact captures the typical way we design a gaming AI: record some information and update the information by an algorithm after the opponent moves. In general,

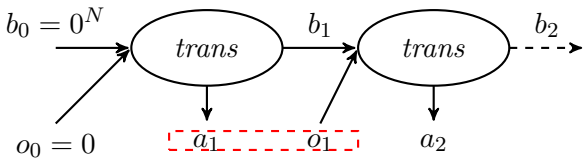

Figure 1: An illustration of the function *trans*. The red dashed rectangle indicates the action profile of the first round.

most of the AI does not sweep through the history every time when making a move, which is costly and typically unaffordable.

In the time-restricted case, we studied how much time resource is needed to find or implement a best response of a time-restricted strategy. In this case, since we focus on space resources, it is natural to study how much space resource is needed for the same tasks of a space-restricted strategy.

To do this, we should first define how to measure the amount of memory used by a strategy in our model. For an agent to work with a space-restricted strategy, it needs space to evaluate the function *trans* and store the description of the function *trans* itself. So it is natural to define the amount of space used by a space-restricted strategy to be the number of storage bits needed to evaluate the function *trans* plus the number of bits needed to specify the function *trans*.

In the following part of this section, we study the amount of space needed for a general TM to compute the best response for a space-restricted strategy and to implement the best response via a space-restricted strategy. We consider two cases. In the first case, we require *trans* to be efficiently computable. This leads to the *circuit strategy model*. In the second case, we drop the computation requirement and consider the *inplace strategy model*.

4.1 Circuit Strategy

In this section we consider the case that we require that function *trans* can be efficiently computed. In this case, since the number of information bits N is fixed, we can then represent *trans* as a polynomial size circuit. We can benefit from a circuit representation as a polynomial size circuit will always halt and can always be efficiently computed, which makes the analysis easier.

Definition 10. *An N-bits-circuit strategy is a boolean circuit C which has $N+1$ input gate and $N+1$ output gate. The size of a circuit strategy is the number of gates in C plus the binary description size of C.*

We include the number of gates in the size of C as for each gate we need one bit to record its output in order to evaluate the circuit C.

4.2 Inplace Strategy

In this section, we drop the computation efficiency requirement and consider the so-called inplace strategy defined as follows.

Definition 11. *An N-bits-inplace strategy is a TM M which runs on input of $N+1$ bits, always halts, and uses only $N+1$ bits of space, returns the content of tape as output when it halts. The size of an inplace strategy is N plus the binary description size of M.*

The name "inplace strategy" is due to the fact that the strategy is implemented by an "inplace" TM, which does not use any extra space other than the input tape itself. Note that in this case, it does not matter whether M accepts or rejects.

As dealing with time-restricted strategies, we are still faced with the same problem: how do we know M is an inplace strategy of N bits? We address this problem by a similar manner as we have done for time-restricted strategies.

Let M be a TM, define M_I as a TM such that it runs M on input string x of length N. If M tries to access tape cells outside the N input bits or does not halt after $Q \cdot N2^N$ steps where Q is the number of the states in M, then M_I halts. M_I returns the content on the tape when it halts.

Lemma 3. *If* TM M *is an* $(N-1)$*-bits-inplace strategy, the output of* M_I *is the same as the output of* M *for every input of length* N*.*

Proof. Since the used tape is of size at most N, there are at most $Q \cdot N2^N$ configurations for M. Suppose by contradiction that, we have ran M for $Q \cdot N2^N$ steps and it still doesn't halt. Then, there are at most $(Q-1)N2^N$ configurations that are not in accepting or rejecting state. By pigeonhole principle, it must visit some configuration twice. In other words, it will loop forever, which leads to a contradiction. □

4.3 Complexity of Computing Best Response

Similar to what we have done for time-restricted cases, to study the space complexity for computing best response against a circuit strategy or an inplace strategy, we first study the decision version, and then use the decision version as a subroutine to find the best response.

We first introduce two sets of languages BRCT and BRIP, which are decision versions of finding best response against circuit strategy and inplace strategy, respectively.

Definition 12. BRCT $= \{\langle C, n, k\rangle\}$ *such that there exists a strategy can yield at least utility* k *against circuit strategy* C *in the game* G^n*.* BRIP $= \{\langle M, 1^N, n, k\rangle\}$ *such that there exists a strategy can yield at least utility* k *against an inplace strategy* M_I *with* N *information bits in the game* G^n*.*

Similar to the time-restricted case, once we have oracle access to the decision version, computing the best response can also be done by an algorithm similar to that of Theorem 1 in polynomial space, which implies the following lemma.

Lemma 4. *For a given inplace (circuit) strategy* M*, we can find the best response at each round in* $\mathsf{PSPACE}^{\mathsf{BRIP}}$ *or* $\mathsf{PSPACE}^{\mathsf{BRCT}}$*.*

Lemma 4 suggests that, in order to study the complexity of finding best response against a circuit strategy or an inplace strategy, it suffices to study the decision version of the problem, which was given in the following theorems.

Theorem 2. BRIP *is* PSPACE*-complete.*

Proof. A simple NPSPACE algorithm is to enumerate all possible action sequences and then simulate the inplace strategy of the opponent to check whether each action sequence can gain utility at least k. As NPSPACE = PSPACE [19], BRIP is in PSPACE. Also, simulating an inplace TM to get its output in first round is already PSPACE*-hard*. As a result, BRIP is PSPACE*-complete*. □

An NPSPACE algorithm is to enumerate all possible action sequences and simulate the inplace strategy to check whether it can gain utility at least k. As NPSPACE = PSPACE[19], BRIP $\in$ PSPACE. The hardness part can be found in appendix.

Theorem 3. BRCT *is* PSPACE*-complete.*

Similar to Theorem 2, constructing a PSPACE algorithm is easy. The non-trivial part is to show that BRCT is PSPACE-*complete*. In order to prove that, we first introduce some necessary notations.

Let the alphabet of a TM be $\{0,1\}$ and the unused cells of the tape be filled with #. Define the configuration of a TM as follows.

Definition 13. *A configuration u of a* TM M *is a triple $(q, pos, content) \in Q \times \mathbb{N} \times \{0,1\}^*$, where Q is the set of states of M, q is the current state, pos is the location of the head pointer and content is the contents of all non-blank cells of the tape.*

For a NTM M, define $\text{next}_c(M, u)$ to be the configuration after running M for one step on configuration u with the nondeterministic choice to be c. If M have already halted on u, define $\text{next}_c(u) = u$.

The intuition of the proof comes as follows. First, as PSPACE = NPSPACE, it is sufficient to prove BRCT is NPSPACE-*complete*.

For a NPSPACE TM M, its configurations can be described by a polynomial number of bits. We construct a circuit strategy C whose information bits $u \in \{0,1\}^*$ describe a configuration of M. If u is not halted, C defects and treats the opponent's action as the nondeterministic choice c and outputs $\text{next}_c(M, u)$ as information bits. Otherwise, C outputs u as information bits, cooperates if u is in an accepting state and defects if u is in a rejecting state.

To test whether M accepts a input string s, note that for a sufficient long running, if there is such a sequence of nondeterministic choices that lead M to an accepting state (which means M accepts s), then C will always cooperate after that, so we can gain a relatively high utility. Otherwise, C will always defect, which causes a low utility. Now we are ready to state the formal proof.

Proof. As we have shown, BRCT is in PSPACE. Thus, in order to show BRCT is PSPACE-*complete*, it is sufficient to show that BRCT is NPSPACE-*hard*, as PSPACE = NPSPACE.

We prove this part by showing for *any* language $L \in$ NPSPACE, $L \leq_p$ BRCT.

For a language $L \in$ NPSPACE, there exists a polynomial $f : \mathbb{N} \to \mathbb{N}$ and a NTM M can decide it using $f(n)$ space and $2^{f(n)}$ time on input string of length n.

Now let us construct the circuit that decodes the configurations of M. W.l.o.g., we assume there is only 3 possible alphabet $\{0, 1, \#\}$. For each tape cell c, there are 2 possibilities:

- The head pointer is not on c.
- The head pointer is on c and the current state of M is q.

Now we define a new alphabet $\Gamma = \{0, 1, \#\} \cup (\{0, 1, \#\} \times Q)$ where Q is the set of states of M. Note that a configuration of M on input of size n can be encoded as string $u \in \Gamma^{f(n)}$ in a natural way, i.e., for each tape cell without the head pointer on, we use a symbol in $\{0, 1, \#\}$ to encode it; for the tape cell with the head pointer on, we use a symbol in $\{0, 1, \#\} \times Q$ to encode the cell's content and the current state q of M.

For a configuration (see Definition 13) u of a TM M, we define variable $x_{i,j}$ to indicate whether u_i is equal to Γ_j where u_i denotes the i-th character of the configuration u and Γ_j denotes the j-th character in Γ. Thus, the variables $x_{1,1}, \ldots, x_{f(n),|\Gamma|}$ encode a configuration of M.

Denote u to be a configuration of a TM M. We construct a linear circuit C which takes $c, x_{1,1}, \ldots, x_{f(n),|\Gamma|}$ as input and outputs $a, y_{1,1}, \ldots, y_{f(n),|\Gamma|}$. $x_{1,1}, \ldots, x_{f(n),|\Gamma|}$ encode the configuration u while $y_{1,1}, \ldots, y_{f(n),|\Gamma|}$ encode the configuration $\text{next}_c(u)$. Meanwhile, the output bit a equals

zero if and only if u is in an accepting state. Here we omit the tedious details of the construction the circuit C.

To summarize, C is a circuit with size $O(f(n))$ and treats the information bits as a configuration u of M, the opponent's action o as the nondeterministic choice c. It cooperates if and only if u is in an accepting state and outputs the configuration of $\mathrm{next_c(u)}$.

Let a be a binary string of length $|a|$. To decide whether $a \in L$ or not, we use the circuit strategy C to simulate the running of M on a.

Consider the instance $x = \langle C, 10 \cdot 2^{f(|a|)}, 20 \cdot 2^{f(|a|)} \rangle$. If $a \in L$, there exists a sequence of nondeterministic choices $u \in \{0,1\}^{2^{f(|a|)}}$ which leads C to an accepting configuration. Thus, the best response has a utility of at least $45 \cdot 2^{f(|a|)}$, which means $x \in \mathsf{BRCT}$. If $a \notin L$, C will always defect and the maximum utility we can only gain is at most $10 \cdot 2^{f(|a|)}$, which means $x \notin \mathsf{BRCT}$. This concludes the theorem. □

The above results demonstrate that computing a best response against a space-restricted strategy can be done in polynomial space, in contrary to the time-restricted case, where it is NP-*complete* to compute the best response.

4.4 Complexity of Implementing Best Response

Now, we study the space complexity for implementing a best response of a particular space-restricted strategy, which is equivalent to the question what is the smallest possible size among all best responses.

From previous sections, the algorithm for computing the best response uses polynomial space. Then the natural question is whether it can be done in linear space? Notice that as the input string of the algorithm is $\langle M, n, k \rangle$ which has length $|M| + \log n + \log k = |M| + O(\log n)$, thus we when say polynomial/linear space algorithm, we refer to a polynomial/linear function of $|M| + O(\log n)$.

We have the following theorem which demonstrates that it is impossible to have a linear space best response against an inplace strategy under reasonable complexity conjecture.

Theorem 4. *Unless* $\mathsf{DSPACE}(n) = \mathsf{NSPACE}(n)$*, there does not exist a constant* T *such that any inplace strategy of size* S *in super game* G^n *have a best response implemented by an inplace strategy with size smaller than* $T \cdot (S + \log n)$*.*

The intuition here is to construct an agent that simulates the behavior of a NTM on a specific input by treating the opponent's actions as the nondeterministic choices. The agent will cooperate only if it is in an accepting state. Thus, the best response strategy will output a sequence of nondeterministic choices which makes the NTM end in an accepting state. If the best response can be implemented in linear space, we can then construct a DTM which enumerates all possible inplace strategies with linear size to find the best response. By finding the best response, we can actually get the nondeterministic choices of the NTM, and thus can simulate the running of a NTM on a DTM, still by using linear space, which contradicts our assumption that $\mathsf{DSPACE}(n) \neq \mathsf{NSPACE}(n)$. Now we present the detailed proof.

Proof. Suppose by contradiction that there exists such a constant T.

Let a language $L \in \mathsf{NDSPACE}(n)$, it follows that there exists a constant c and a NTM M that decides L using $c \cdot n$ space and $2^{c \cdot n}$ time on input of size n. We encode M's configuration in the same way as in the proof of Theorem 3. Note we can simulate the circuit strategy C described in the proof of Theorem 3 by an inplace strategy U with only additional linear space. Thus, the number of information bits used by U is still linear in n.

Given a string $a \in \{0,1\}^n$, to answer whether $a \in L$ or not, we use U to simulates the running of M on a. U can be described by linear number bits to encode the string a and uses linear number of information bits, so the size of U is linear in n.

Consider the best response against U in the repeated game $G^{10\cdot 2^{c\cdot n}}$. Recall that U will cooperate only if it is in an accepting state. Thus, if $a \in L$, the best response will produce the nondeterministic choices to make M end in an accepting states. In addition, according to the assumption, the best response can be implemented by an inplace strategies V with size linear in n, as $\log(10 \cdot 2^{c\cdot n}) = O(n)$ and the size of U is linear in n. Thus, in order to judge whether $a \in L$ or not, we first enumerate all possible linear size inplace strategy V and simulate the game to find a sequence of nondeterministic choices to make M end in an accepting state. This can be done by a linear size DTM. Hence, $\mathsf{NDSPACE}(n) \subset \mathsf{DSPACE}(n)$, which leads to a contradiction. □

The taken-away message of Theorem 4 is that in general, implementing a best response of a particular strategy need much more space than the strategy itself.

5 Nash Equilibria via Restricted Turing Machine

In this section, we study the case when both player are using restricted Turing machines strategies and this is a common knowledge. We are going to study infinitely repeated game from now on. For simplicity of analysis, we use the standard limit of mean as the utility notion.

Definition 14. *In an infinitely super game G^∞, denote s_i as the strategy of player i. The utility for player i is $U_i(s_1, s_2) = \liminf_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} u_i(a_{t,1}, a_{t,2})$, where $a_{t,i}$ is the action of the player i at the t-th round.*

Suppose s is a strategy, Let $\mathcal{S}$ be the set of all possible strategies in G^∞, denote $\mathsf{BR}(s) = \sup\{U_2(s,t) \mid t \in \mathcal{S}\}$. We say a strategy t is a best response of s if $U_2(s,t) = \mathsf{BR}(s)$. Note that it is possible that there is no best response for s.

Suppose s is a strategy, and $\mathcal{C}$ is a complexity class, denote $\mathsf{BR}_{\mathcal{C}}(s) = \sup\{U_2(s,t) \mid \text{t is a } \mathcal{C}\text{-strategy}\}$. We say t is a $\mathcal{C}$-best response of s if $U_2(s,t) = \mathsf{BR}_{\mathcal{C}}(s)$. Based on these notations, we are now ready to define $\mathcal{C}$-Nash Equilibrium ($\mathcal{C}$-NE).

Definition 15. *A $\mathcal{C}$-NE of an infinitely super game G^∞ is a pair of strategy (s_1, s_2) such that both s_1 and s_2 are $\mathcal{C}$-strategies, and none of them can gain higher utility by deviating to another $\mathcal{C}$-strategy.*

Our goal now is to investigate how does such restriction affect the set of NE. At first glance, such a restriction will disqualify some old NEs. Surprisingly, such restriction will also produce some new NEs.

Lemma 5. *There exists a TM-NE that is not a NE, and a NE which is not a TM-NE.*

Lemma 6. *There exists a P-NE that is not a TM-NE, and a TM-NE which is not a P-NE.*

Moreover, we have some stronger results summarized in the following two theorems. We say a $f : \mathbb{N} \to \mathbb{N}$ is a reasonable function, if f is a strictly increasing time constructible function such that $f(0) > 0$.

Theorem 5. *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two reasonable functions, such that $f(n)\log f(n) \in o(g(n))$ and $f(n) \in \Omega(n \log n)$. There exists a $\mathsf{DTIME}(f(n))$-NE which is not a $\mathsf{DTIME}(g(n))$-NE, and a $\mathsf{DTIME}(g(n))$-NE which is not a $\mathsf{DTIME}(f(n))$-NE.*

Theorem 6. *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two reasonable functions, such that $f(n) \in o(g(n))$ and $f(n) \in \Omega(\log n)$. There exists a $\mathsf{DSPACE}(f(n))$-NE which is not a $\mathsf{DSPACE}(g(n))$-NE, and a $\mathsf{DSPACE}(g(n))$-NE which is not a $\mathsf{DSPACE}(f(n))$-NE.*

We sketch the essence of the proofs here. Full proof can be found in the following section. Let $\mathcal{C}, \mathcal{D}$ be two complexity classes such that $\mathcal{C} \subset \mathcal{D}$. Each of our results has two parts: there exists a $\mathcal{D}$-NE which is not a $\mathcal{C}$-NE, and there exists a $\mathcal{C}$-NE which is not a $\mathcal{D}$-NE.

We first construct a $\mathcal{C}$-strategy s_1 that s_1 has a $\mathcal{D}$-strategy as the best response but no $\mathcal{C}$-strategy as the best response. For this purpose, we construct a hard problem $\mathcal{P}$. And in some specific rounds, s_1 treats the opponent's action as the answer to "what is the value of $\mathcal{P}(x)$?" where x is dependent on the current round number. s_1 will check whether the opponent's answer is right in later rounds. Once s_1 finds an incorrect answer, s_1 defects forever, otherwise s_1 cooperates. Thus, in order to be the best response of s_1, the opponent should be able to solve all questions correctly. Then we can construct $\mathcal{P}$ in a way that no machine of complexity $\mathcal{C}$ can compute it, but some machine in complexity $\mathcal{D}$ can.

To prove the first part, we further construct a $\mathcal{D}$-strategy s_2 that s_1 and s_2 together constitute a NE. Obviously they constitute a $\mathcal{D}$-NE. And as s_1 has no $\mathcal{C}$-strategy as the best response, so they are not $\mathcal{C}$-NE.

To prove the second part, we construct a hybrid strategy t such that it asks the opponent to make a two-decision choice at the first round. If the opponent choose the first choice, t then acts like a strategy t_1 which is easy to make best response and $\mathsf{BR}(t_1) < \mathsf{BR}(s_1)$, and for the second choice, t will then act like strategy s_1.

Consider another strategy v which chooses the first choice and then behaves the same as t_1's best response. t and v forms a $\mathcal{C}$-NE if we construct them carefully. But they does not form $\mathcal{D}$-NE, as v can make profitable deviation by choosing the second choice and acts like s_1's best response.

5.1 Detailed Proofs of Lemma 5, Lemma 6, Theorem 5 and Theorem 6

In the following we provide detailed proofs for lemmas and theorems stated in the previous section.

5.1.1 Notations

We begin with some notations. We first define the following simple alternating strategy, which will be used in our proofs.

Definition 16. ALT *is a strategy which cooperates at odd rounds and defects at even rounds. In addition, if it finds the opponent defects in odd rounds or cooperates at even rounds. It will then start defecting forever.*

ALT has the following properties which are important to us.

Lemma 7. ALT *is a P-strategy, more precisely, a $\mathsf{DTIME}(n)$-strategy. And ALT, ALT is a pair of NE in which every one get utility 2.*

Proof. Note that we have a simple automaton that recognizes ALT, so there is a $\mathsf{DTIME}(n)$ Turing machine which can implement it. Meanwhile, the best response of ALT is obviously ALT itself. □

We also need the following two constructions, which can construct complex strategy from simpler ones.

For two strategies s_1, s_2. We denote $\mathsf{Choice}(s_1, s_2)$ as the following strategy: it treats the opponent's action in the first round as a "choice"; if the opponent cooperates, then it ignores what

happened in the first round and acts like s_1; and if the opponent defects, it acts like s_2 in the same way.

Similarly, for a strategy s and an integer $a \in \{0, 1\}$, we use $\mathsf{Con}(a, s)$ to denote the strategy that makes action a at the first round (recall that 1 represents cooperate and 0 represents defect), and then acts like strategy s which ignores what happens in the first round.

Finally, we say a complexity class $\mathcal{C}$ is reasonable, if for any two $\mathcal{C}$-strategies s_1, s_2 and an integer $a \in \{0, 1\}$, both $\mathsf{Choice}(s_1, s_2)$ and $\mathsf{Con}(a, s)$ are $\mathcal{C}$-strategies.

5.1.2 A Strategy Framework

We set up a strategy framework first.

The basic idea is that we construct a strategy that forces the opponent to correctly solve some hard problems and check the answer later. As the computation power we have grows with the input length of the TM, eventually we will be able to check the answer's validity even if we can't do that immediately.

Define a promise problem $\mathcal{P}$ to be a function $\mathbb{N} \to \{0, 1, \star\}$. Note $\mathcal{P}(x) = \star$ means that $\mathcal{P}(x)$'s value is undefined. We say an answer $a \in \{0, 1\}$ to $\mathcal{P}(x)$ is correct iff $\mathcal{P}(x) = \star$ or $a = \mathcal{P}(x)$.

We also need to fix a bijection τ between positive integers ($\mathbb{N}$) and all unary strings in $\{1\}^*$: we map $i \in \mathbb{N}$ to 1^i (a string consists of i ones). So when we use an integer i to denote an input to a TM, we mean the actual input is 1^i. We also use $|i|$ to denote the length of the i-th unary string 1^i, which is simply just i.

We say a sequence $\{t_i\}_{i=1}^{+\infty}$ is *checking sequence*, if it consists of positive integers and is strictly increasing. Moreover, we require that $\lim_{i\to\infty} \frac{i}{t_i} \leq 0.1$, i.e., the distribution of the t_i's is sparse.

Fix a promise problem $\mathcal{P}$ and a checking sequence $\{t_i\}_{i=1}^{+\infty}$, we say a strategy $s_{\mathcal{P}}$ is a $(\mathcal{P}, \{t_i\})$-checking strategy, if it satisfies the following conditions.

- For each i, we call the t_i-th round the i-th *checking* round; and we say a round is *non-checking* if it is not in $\{t_i\}_{i=1}^{+\infty}$.
- At the i-th checking round, $s_{\mathcal{P}}$ treats the opponent's action as an answer to $\mathcal{P}(i)$.
- If the opponent defects in any non-checking rounds, $s_{\mathcal{P}}$ will start defecting forever.
- For each i, if the opponent's answer to $\mathcal{P}(i)$ in the i-th checking round is wrong. There always exists a later round such that $s_{\mathcal{P}}$ discovers this mistake, and starts defecting forever.

Remark 1. *We can see in order to specify a $(\mathcal{P}, \{t_i\})$-checking strategy $s_{\mathcal{P}}$, one only need to show how $s_{\mathcal{P}}$ implements the "lazy checking" system required in the last condition.*

For a $(\mathcal{P}, \{t_i\})$-checking strategy $s_{\mathcal{P}}$, we can construct its best response, denoted by $t_{\mathcal{P}}$, as follows: $t_{\mathcal{P}}$ always cooperates in non-checking rounds, and in the i-th checking round, if $\mathcal{P}(i) \neq \star$, it acts according to $\mathcal{P}(i)$, otherwise $\mathcal{P}(i) = \star$ and it defects. Moreover, $t_{\mathcal{P}}$ starts defecting forever whenever it finds its opponent defect.

We have following useful properties regarding the game between $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$.

Lemma 8. *$s_{\mathcal{P}}$ and $t_{\mathcal{P}}$ forms a NE, and they both gain a utility of at least 2.7.*

For any strategy u which fails to answer $\mathcal{P}(i)$ correctly in a checking round, u can only get a utility of 1 in the game between $s_{\mathcal{P}}$.

Proof. In the game between $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$, as $t_{\mathcal{P}}$ answers all questions in the checking-rounds correctly, $s_{\mathcal{P}}$ always cooperates. Therefore, they both always cooperate in all non-checking rounds. Since there are at least a $1 - 0.1 \geq 0.9$ fraction of non-checking rounds, they both gain a utility of at least $3 \cdot 0.9 = 2.7$.

Then it is easy to see that $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$ forms a NE: $s_{\mathcal{P}}$ can not deviate profitably, as if it defects in any round, $t_{\mathcal{P}}$ will start defecting forever and the utility will drop to 1; similarly, one can argue that $t_{\mathcal{P}}$ can not deviate profitably neither.

Finally, if a strategy u fails to answer $\mathcal{P}(i)$ correctly at the i-th checking round, by the properties of $s_{\mathcal{P}}$, $s_{\mathcal{P}}$ will eventually start defecting forever, and consequently u can only get a utility of 1. □

Now, we are going to state our framework lemma, which enables us to prove all our results in a unified way.

Lemma 9. *Let $\mathcal{C}, \mathcal{D}$ be two reasonable complexity classes*[1] *such that $\mathcal{C} \subset \mathcal{D}$, $\mathcal{P}$ be a promise problem and $\{t_i\}$ be a checking sequence. Suppose the following conditions holds:*

- *There exists a $(\mathcal{P}, \{t_i\})$-checking, $\mathcal{C}$-strategy $s_{\mathcal{P}}$.*
- *In the game with $s_{\mathcal{P}}$, no $\mathcal{C}$-strategy can answer all questions to $\mathcal{P}$ correctly.*
- *$t_{\mathcal{P}}$ is a $\mathcal{D}$-strategy.*
- ALT *is a $\mathcal{C}$-strategy.*

Then there exists a $\mathcal{C}$-NE which is not a $\mathcal{D}$-NE and a $\mathcal{D}$-NE which is not a $\mathcal{C}$-NE.

Proof. We prove the two parts of the claim separately.

There exists a $\mathcal{D}$-NE which is not a $\mathcal{C}$-NE:

We prove the latter part of the claim first as it is relatively easier. Consider the strategy pair $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$. We know they form an NE, since they are both $\mathcal{D}$-strategies ($\mathcal{C} \subset \mathcal{D}$), the also form a $\mathcal{D}$-NE. However, $t_{\mathcal{P}}$ is not a $\mathcal{C}$-strategy, as it answers all questions correctly. Therefore, they are not a $\mathcal{C}$-NE, and this proves the latter part of the claim.

There exists a $\mathcal{C}$-NE which is not a $\mathcal{D}$-NE:

The first part of the claim requires a bit more work. The idea here is asking the opponent to make a choice at the first round, one for $\mathcal{C}$ and one for $\mathcal{D}$, $\mathcal{D}$-strategy can choose the second option and get a higher utility, while $\mathcal{C}$-strategy can not.

Now, consider the strategy pair $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$ and $\mathsf{Con}(1, \mathsf{ALT})$. As both $s_{\mathcal{P}}$ and ALT are $\mathcal{C}$-strategies, and $\mathcal{C}$ is reasonable, $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$ and $\mathsf{Con}(1, \mathsf{ALT})$ are both $\mathcal{C}$-strategies too.

Then we verify that they constitute a $\mathcal{C}$-NE:

It is easy to see $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$ is a best response of $\mathsf{Con}(1, \mathsf{ALT})$, as it obtains a utility 2, which is the maximum possible against ALT or $\mathsf{Con}(1, \mathsf{ALT})$.

Now we argue that $\mathsf{Con}(1, \mathsf{ALT})$ is a $\mathcal{C}$-best response for $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$. For any $\mathcal{C}$-strategy t, if t defects in the first round, which means it has to play against $s_{\mathcal{P}}$, but as the condition indicates, no $\mathcal{C}$-strategy can answer all questions to $\mathcal{P}$ correctly in the game with $s_{\mathcal{P}}$, which means $s_{\mathcal{P}}$ will eventually start defecting forever and t can only get a utility of 1; but if t cooperates in the first round, clearly it should then acts like ALT, thus $\mathsf{Con}(1, \mathsf{ALT})$ is a $\mathcal{C}$-best response for $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$.

Putting them together, we can see $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$ and $\mathsf{Con}(1, \mathsf{ALT})$ form a $\mathcal{C}$-NE.

However, since $\mathsf{Con}(1, \mathsf{ALT})$ can gain a profit of 2.7 by deviating to a $\mathcal{D}$-strategy $\mathsf{Con}(0, t_{\mathcal{P}})$, $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$ and $\mathsf{Con}(1, \mathsf{ALT})$ are not a $\mathcal{D}$-NE, which completes the proof. □

[1] In fact, the proof only requires class $\mathcal{C}$ to be reasonable.

5.1.3 Proof of Lemma 5

Then we are ready to prove Lemma 5, we restate it here for convenience.

Lemma 5 (restated) *There exists a* TM*-NE that is not a NE, and a NE which is not a* TM*-NE.*

Proof. We apply Lemma 9 with $\mathcal{C} = \mathsf{TM}$ (which means all computable languages) and $\mathcal{D} = \mathsf{ALL}$ (which means all languages).

The hard problem H_{TM} and checking sequence $\{t_i\}$.

We first set up the hard problem $\mathcal{P} = \mathsf{H}_{\mathsf{TM}}$, which is a simple variant of the famous *halting problem*, let TM_x denote the TM described by integer x. If TM_x halts on the input x, we define $\mathsf{H}_{\mathsf{TM}}(x)$ to be its output, otherwise we set $\mathsf{H}_{\mathsf{TM}}(x) = \star$.

And we set the checking sequence as$\{t_i = 10i\}_{i=1}^{+\infty}$.

By standard diagonalization method (see e.g. [20]), we can show no TM which always halts can solve H_{TM} correctly.

Construction and Analysis of $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$.

Now we construct our $(\mathsf{H}_{\mathsf{TM}}, \{t_i\})$ checking strategy $s_{\mathcal{P}}$. By Remark 1, we only need to specify how $s_{\mathcal{P}}$ checks the answer to H_{TM} given by its opponent.

$s_{\mathcal{P}}$ works as follows: on the n-th round, for each i such that $t_i < n$, it runs TM_i on input i for n steps; if TM_i halts, $s_{\mathcal{P}}$ compares TM_i's output with the answer given by its opponent (which are recorded in the history), and will start defecting forever if they are different.

We can see that $s_{\mathcal{P}}$ is indeed a P-strategy (hence also a TM-strategy). Furthermore, when the answer a given by its opponent in the i-th checking round is not correct, which means TM_i halts on input i with an output x, and $x \neq a$. Suppose TM_i have run T steps on input i, then we can see $s_{\mathcal{P}}$ will find out this mistake during its checking before or at the $\max\{T, i+1\}$-th round.

Also, $t_{\mathcal{P}}$ is clearly a ALL-strategy (ALL contains all languages).

No TM-strategy can answer all questions from $s_{\mathcal{P}}$.

We will show that if such a TM-strategy u exists, then one can construct a TM solving $\mathcal{P}$ correctly, which renders a contradiction.

The TM M is rather simple: in order to answer $\mathcal{P}(i)$, it simply simulates the game between $s_{\mathcal{P}}$ and u up to the t_i-th round, and output the action by u on t_i-th round. By the property of u, we can see M solves $\mathcal{P}$ correctly, furthermore, since u and $s_{\mathcal{P}}$ are both TM-strategy, M is a TM which always halts, contradiction.

Applying Lemma 9.

Finally, we apply Lemma 9 with $\mathcal{C} = \mathsf{TM}$ and $\mathcal{D} = \mathsf{ALL}$, which completes the proof of Lemma 5. □

5.1.4 Proof of Theorem 5 and Theorem 6

Next we prove Theorem 5, which is a bit more technical, we restate it here for convenience. Recall that a $f : \mathbb{N} \to \mathbb{N}$ is a reasonable function, if f is a strictly increasing time constructible function such that $f(0) > 0$.

Theorem 5 (restated) *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two reasonable functions, such that $f(n) \log f(n) \in o(g(n))$ and $f(n) \in \Omega(n \log n)$. There exists a $\mathsf{DTIME}(f(n))$-NE which is not a $\mathsf{DTIME}(g(n))$-NE, and a $\mathsf{DTIME}(g(n))$-NE which is not a $\mathsf{DTIME}(f(n))$-NE.*

Proof. Similar to the proof of Lemma 5, we are going to apply Lemma 9 with $\mathcal{C} = \mathsf{DTIME}(f(n))$ and $\mathcal{D} = \mathsf{DTIME}(g(n))$.

The checking sequence $\{t_i\}$. We are going to construct the checking sequence $\{t_i\}$ as follows. Let $t_1 = 1$, and for each $i > 1$, we set $t_i = t_{i-1} + g(2t_{i-1}) + 10$.

Since $t_i - t_{i-1} \geq 10$ for all $i > 1$, we can see $\lim_{i \to +\infty} \frac{i}{t_i} \leq 0.1$ as required. And note that g is a positive strictly increasing function, we have $t_i \geq t_{i-1} + g(2t_{i-1}) \geq 3t_{i-1}$.

Moreover, it is easy to see the function $\mathbf{t} : i \to t_i$ is time constructible as g is time constructible.

Scaling the input length.

Note that at the i-th checking round, the input length (which is just the length of history) is indeed $2t_i - 2$, so we let $F(n) = f(2t_n - 2)$ and $G(n) = g(2t_n - 2)$. Clearly, $F(n) \log F(n)$ is still $o(G(n))$.

From the definition of t_i, we have $F(i) = f(2t_i - 2) \geq 2t_i - 2 \geq 2g(2t_{i-1}) \geq 2G(i-1)$; similarly, $G(i) = g(2t_i - 2) \geq 2t_i - 2 \geq 2g(2t_{i-1}) \geq 2G(i-1)$.

In addition, as $\mathbf{t}$ is time constructible, both functions $F(n)$ and $G(n)$ are time constructible.

The "halt before $h(x)$ steps" problem H_h.

In order to make use our framework, we first construct a hard problem which will somehow separate $\mathsf{DTIME}(F(n))$ and $\mathsf{DTIME}(G(n))$ in the same spirit as the time hierarchy theorem [8].

Now, given a time constructible function h, we define the "halt before $h(x)$ steps" problem, denoted as H_h, as follows: for an integer x, if TM_x halts on input x after at most $h(|x|)$ steps, $\mathsf{H}_h(x)$ is defined to be the output of TM_x; if TM_x doesn't halt after $h(|x|)$ steps, we let $\mathsf{H}_f(x) = \star$ instead.

No $o(h(n))$-time TM can solve H_h.

Now we show that a TM runs in time $o(h(n))$ can not solve all H_h. Suppose there exists such a TM M, we flip M's output to get another TM M_{flip}. Clearly M_{flip} also runs in $o(f(n))$ steps, which means there exists an integer N such that for all input size $n > N$, M_{flip} runs less than $f(n)$ steps.

By adding some dummy states in M_{flip}, we can let M_{flip} have size $|M_{\mathsf{flip}}| > N$. Then we have $\mathsf{H}_h(M_{\mathsf{flip}}) = M_{\mathsf{flip}}(M_{\mathsf{flip}}) = 1 - M(M_{\mathsf{flip}}) = 1 - \mathsf{H}_h(M_{\mathsf{flip}})$ as M_{flip} halts after $f(|M_{\mathsf{flip}}|)$ steps and $M(M_{\mathsf{flip}}) = \mathsf{H}_h(M_{\mathsf{flip}})$ by assumption, contradiction.

Our hard problem $\mathcal{P}$.

Recall that $F(n) \log F(n)$ is $o(G(n))$, we are going to prove $F(n) = o(G(n)/\log G(n))$.

Note that when $F(n)^2 \leq G(n)$, we have $F(n) \leq \sqrt{G(n)}$ and $F(n) \log G(n) \leq \sqrt{G(n)} \log G(n)$; when $F(n)^2 > G(n)$, we have $\log F(n) \geq \frac{1}{2} \cdot \log G(n)$ and $F(n) \log G(n) \leq 2F(n) \log F(n)$.

Putting them together, for all n we have

$$\begin{aligned} F(n) \log G(n) &\leq \max\{\sqrt{G(n)} \log G(n), 2F(n) \log F(n)\} \\ &= o(G(n)), \end{aligned}$$

which means $F(n) = o(G(n)/\log G(n))$.

Now, we set $h(n) := G(n)/\log G(n)$, and define our hard problem $\mathcal{P} = \mathsf{H}_h$.

Construction and Analysis of $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$.

We construct the $(\mathcal{P}, \{t_i\})$-checking strategy $s_{\mathcal{P}}$ similarly as in the proof of Lemma 5. By Remark 1, we only need to specify how $s_{\mathcal{P}}$ checks the answer to $\mathcal{P} = \mathsf{H}_h$ given by its opponent.

$s_{\mathcal{P}}$ works as follows: on the n-th round, for each i such that $t_i < n$, it runs TM_i on input i for $\min\{\sqrt{n}, G(i)/\log G(i)\}$ steps; if TM_i halts, $s_{\mathcal{P}}$ compares TM_i's output with the answer given by its opponent (which are recorded in the history), and will start defecting forever if they are different.

Now we show $s_{\mathcal{P}}$ is indeed a $\mathsf{DTIME}(n \log n)$-strategy (hence also a $\mathsf{DTIME}(f(n))$-strategy). $s_{\mathcal{P}}$ first scan the input to count its length, which takes $O(n \log n)$ time. After that, since $t_i \geq 3t_{i-1}$ for $i > 1$, there are at most $\log n$ values of t_i's to verify. Each verification needs at most $O(\sqrt{n} \cdot \log n)$ time, so they together takes at most $O(\sqrt{n} \log^2 n) = O(n \log n)$ time.

Moreover, when the answer a given by its opponent in the i-th checking round is not correct, which means TM_i halts on input i with an output x before $h(i)$ steps, and $x \neq a$. Then we can see $s_{\mathcal{P}}$ will find out this mistake during its checking before or at the $\max\{h(i)^2, i+1\}$-th round.

The $t_{\mathcal{P}}$ works by solving each question to H_h correctly. $t_{\mathcal{P}}$ first scan the input to count its length, which takes $O(n \log n)$ time. After that, it just cooperates on non-checking rounds; and on the i-th checking round, it simulates TM_i on input i for $h(i) = G(i)/\log G(i)$ steps to figure out the answer for $\mathsf{H}_h(i)$, which needs $O(h(i) \log h(i)) = O(G(i)) = O(g(2t_i - 2))$ time, note $2t_i - 2$ is the input length (length of history) at the i-th checking round, hence $t_{\mathcal{P}}$ is a $\mathsf{DTIME}(g(n))$-strategy.

No $\mathsf{DTIME}(f(n))$-strategy can answer all questions from $s_{\mathcal{P}}$.

Finally, we show that if there is a $\mathsf{DTIME}(f(n))$-strategy u can answer all questions from $s_{\mathcal{P}}$, then we can construct a $\mathsf{DTIME}(F(n))$ TM to solve H_h, but since $F(n) = o(h(n))$, it renders a contradiction.

The idea is simple, we just want to simulate u to get its output in the i-th checking round to answer $\mathsf{H}_h(i)$.

To do this, we have to simulate the game between u and $s_{\mathcal{P}}$ up to t_i-th round. However, unlike in Lemma 5, we can not afford a straightforward simulation (which could require $O(t_i \cdot F(i))$ time), and need to make some observations to speed it up.

Our simulation works in two steps, in the first step it prepares the "correct history" of the game between $t_{\mathcal{P}}$ and $s_{\mathcal{P}}$ before the t_i-th round. By the assumption on u, this is identical to the game between u and $s_{\mathcal{P}}$. In the second step it runs the machine of u on the previously constructed history, to output the desired answer.

The preparation is straightforward: since $s_{\mathcal{P}}$ always cooperates; $t_{\mathcal{P}}$ always cooperates on non-checking rounds, and acts according to H_h on checking rounds, so we only have to figure out the answers to $\mathsf{H}_h(t_1), \mathsf{H}_h(t_2), \ldots, \mathsf{H}_h(t_{i-1})$. Evaluating the value of $\mathsf{H}_h(t_j)$ is simply simulating TM_j on input j for $h(j) = G(j)/\log G(j)$ steps, which needs $O(h(j) \log h(j)) = O(G(j))$ time. Therefore, recall that $2G(j) \leq G(j+1)$ for $j \in \{1, \ldots, i-1\}$ and $2G(i-1) \leq F(i)$, the whole preparation takes $O\left(\sum_{j=1}^{i-1} G(j)\right) = O(G(i-1)) = O(F(i))$ time.

Then we simply run u on the constructed history to get its action, by the assumption on u, the action is an answer to $\mathsf{H}_h(i)$, moreover, it takes $O(f(2t_i - 2)) = O(F(i))$ time.

Putting them together, we have an $O(F(i))$ time algorithm for computing $\mathsf{H}_h(i)$, but since $F(i) = o(h(i))$, it renders a contradiction.

Applying Lemma 9.

Finally, we apply Lemma 9 with $\mathcal{C} = \mathsf{DTIME}(f(n))$ and $\mathcal{D} = \mathsf{DTIME}(g(n))$, which completes the proof of Theorem 5.

□

Theorem 6 can be proven in the exactly same way by using the space hierarchy theorem instead. As we don't need extra space factor to simulate a TM machine, we can weaken the assumption to $f(n)$ is $o(g(n))$. In addition, since we need $\Omega(\log n)$ space to count the input length, we have to require $f(n)$ to be $\Omega(\log n)$.

5.1.5 Proof of Lemma 6

Finally, we prove Lemma 6. We restate it here for convenience.

Lemma 6 (restated) *There exists a TM-NE that is not a P-NE, and a P-NE which is not a TM-NE.*

Proof. Let $f(n) = 2^n$ and $g(n) = n^n$. Note that $\mathsf{P} \subset \mathsf{DTIME}(2^n)$, and clearly $f(n)$ and $g(n)$ satisfy the requirements for Theorem 5.

Then we invoke Theorem 5 and Lemma 9 to construct a hard problem $\mathcal{P}$, a checking-sequence $\{t_i\}$, a $(\mathcal{P}, \{t_i\})$-checking strategy $s_{\mathcal{P}}$ and its best response $t_{\mathcal{P}}$ such that:

- $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$ is a $\mathsf{DTIME}(n^n)$-NE, but not a $\mathsf{DTIME}(2^n)$-NE.
- $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$ and $\mathsf{Con}(1, \mathsf{ALT})$ is a $\mathsf{DTIME}(2^n)$-NE, but not a $\mathsf{DTIME}(n^n)$-NE.

Then we are going to show the above strategies also witness Lemma 6.

There exists a TM-NE which is not a P-NE.

Observe that $s_{\mathcal{P}}$ and $t_{\mathcal{P}}$ are clearly both TM-strategies, and they in fact constitute a NE. So they also form a TM-NE. But since $t_{\mathcal{P}}$ is not a P-strategy (from the proof of Lemma 9, it is not even a $\mathsf{DTIME}(2^n)$-strategy), they are not a P-NE.

There exists a P-NE which is not a TM-NE.

Observe that $\mathsf{Choice}(\mathsf{ALT}, s_{\mathcal{P}})$ and $\mathsf{Con}(1, \mathsf{ALT})$ are clearly both P-strategies (in the proof of Theorem 6, $s_{\mathcal{P}}$ is in fact a $O(n \log n)$-time strategy). Since they both can not deviate profitably even to any $\mathsf{DTIME}(2^n)$-strategy, they actually form a P-NE. And clearly they are not a TM-NE, as $\mathsf{Con}(1, \mathsf{ALT})$ can deviate profitably even to a $\mathsf{DTIME}(n^n)$-strategy.

□

6 Generalization to Non-trivial Two Person Games

In the previous sections, we prove several important results concerning time (space) complexity of computing (implementing) the best response of a given strategy in finitely repeated PD games. Also, new Nash Equilibria emerge when computational resource restriction is taken into account in infinitely repeated PD games. But indeed, we choose PD games as our running examples is just for the ease of presentation. In this section, we show that our results in previous sections apply to much more general repeated games.

We need to mention that there exists some trivial games for which the best response is trivial. For example, consider a two person game in which whatever the players do. Their utility is the same. Then clearly any strategy is a best response to a fixed strategy.

To start with, we need to specify our setting. Let $G = \langle S_1, S_2, u_1, u_2 \rangle$ be a two-person normal-form game , S_i be the finite set of actions for player i, $u_i : S_1 \times S_2 \to \mathbb{R}$ be the utility function for player i. For each action $a_2 \in S_2$, let $br_1(a_2) = \max_{a_1 \in S_1} u_1(a_1, a_2)$ to be the best response of action a_2. We always consider finding/implement best response against player 2.

Definition 17. *Let G be a two person game in normal form. If for each $a_2 \in S_2$, $br_1(a_2)$ is the same, then we call it a trivial game.*

Lemma 10. *Let G be a trivial game, let $C = br_1(a_2)$ for any $a_2 \in S_2$. Then the best response against any strategy for player 2 in G^n leads to utility $C \cdot n$.*

Obviously, the utility we can gain is upper bounded by $C\dot{n}$. Meanwhile, we can simly achieve $C \cdot n$ utility by taking the single round best response of player 2 in every round.

We can see that for a trivial game, finding the best response against player 2 is trivial. Conversely, for any non-trivial game H, the results in previous sections remain the same. It would be tedious to restate all the previous results. Instead, we use Theorem 1 as an example, all other theorems can be translated in the same way.

Definition 18. *Let $f : \mathbb{N} \to \mathbb{N}$ be a polynomial time computable function,* $\mathsf{BR}_{f,H} = \{\langle M, 1^n, k\rangle\}$ *such that there exists a strategy that can gain at least utility k against the strategy M_f in the game H^n.*

Since H is non-trivial, there exists two actions $a, b \in S_2$ for player such that $br_1(a) > br_1(b)$. With these two actions, we will be able to construct all those strategies which are used in the proof of Theorem 1. We restrict player 2 to use only those two action. We rename the strategy a as *cooperate* and b as *defect*.

Theorem 7. *There exists a polynomial f such that* $\mathsf{BR}_{f,H}$ *is* NP*-complete. And for every polynomial f,* $\mathsf{BR}_{f,H}$ *is in* NP*.*

Just follow the original proof for Theorem 1 but use the new cooperate (a) and defect (b) actions will give us this result.

7 Future Works

There are some intriguing problems to be explored.

- How do the restrictions on strategies affect the set of NE in finitely repeated game? Particularly, to what extent should we restrict an inplace (circuit) strategy so that cooperation can be sustained?
- For circuit strategy and inplace strategy, what if we impose the so-called simple machine preference (i.e., prefer machines with fewer states)?

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity*. Cambridge University Press, 1 edition, 2009.

[2] Elchanan Ben-porath. The complexity of computing a best response automaton in repeated games with mixed strategies. *Games and Economic Behavior*, 2(1):1–12, 1990.

[3] Ruggiero Cavallo and David C. Parkes. Efficient metadeliberation auctions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 50–56, 2008.

[4] L. Elisa Celis, Anna R. Karlin, Kevin Leyton-Brown, C. Thach Nguyen, and David Robert Martin Thompson. Approximately revenue-maximizing auctions for deliberative agents. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012.

[5] Lijie Chen and Pingzhong Tang. Bounded rationality of restricted turing machines. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1673–1674. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[6] Stephen A Cook. The complexity of theorem-proving procedures. pages 151–158, 1971.

[7] Itzhak Gilboa. The complexity of computing best-response automata in repeated games. *Journal of Economic Theory*, 45(2):342–352, 1988.

[8] Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, pages 285–306, 1965.

[9] Vicki Knoblauch. Computable strategies for repeated prisoner' s dilemma. *Games and Economic Behavior*, 7(3):381–389, 1994.

[10] Guillaume Lacôte. Boundedly complex turing machines play the repeated prisoner's dilemma: some results. Technical report, 2005.

[11] Kate Larson and Tuomas Sandholm. Experiments on deliberation equilibria in auctions. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 394–401, 2004.

[12] Kate Larson and Tuomas Sandholm. Mechanism design and deliberative agents. In *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, pages 650–656, 2005.

[13] Nimrod Megiddo and Avi Wigderson. On play by means of computing machines: preliminary version. pages 259–274, 1986.

[14] Albert R Meyer and Larry J Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. pages 125–129, 1972.

[15] John H Nachbar and William R Zame. Non-computable strategies and discounted repeated games. *Economic theory*, 8(1):103–122, 1996.

[16] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[17] Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, pages 358–366, 1953.

[18] Ariel Rubinstein. Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory*, pages 83–96, 1986.

[19] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

[20] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[21] Pingzhong Tang, Yifeng Teng, Zihe Wang, Shenke Xiao, and Yichong Xu. Computational issues in time-inconsistent planning. In *Proceedings of AAAI*, 2017.

[22] Pingzhong Tang and Hanrui Zhang. Unit-sphere games. *International Journal of Game Theory, to appear*, 2016.

[23] James R Wright and Kevin Leyton-Brown. Beyond equilibrium: Predicting human behavior in normal-form games. In *AAAI*, 2010.

[24] James R Wright and Kevin Leyton-Brown. Behavioral game theoretic models: a bayesian framework for parameter analysis. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 921–930. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[25] Song Zuo and Pingzhong Tang. Optimal machine strategies to commit to in two-person repeated games. In *Proceedings of AAAI*, 2015.

A Preliminaries of Complexity Theory

In order to make our discussion self-contained, we introduce some definitions in the theory of computation. Most definitions can be found in [1].

Definition 19. *A function $f : \mathbb{N} \to \mathbb{N}$ is time constructible if the function that maps the string 1^n to the binary representation of $f(n)$ is computable in time $O(f(n))$.*

Definition 20. *A* TM *M's running time is bounded by a function $f : \mathbb{N} \to \mathbb{N}$ if for any input string of length n, it halts before $f(n)$ steps.*

Definition 21. *A* TM *M's space is bounded by a function $f : \mathbb{N} \to \mathbb{N}$ if for any input string of length n, the number of tape cells scanned during the computation is at most $f(n)$.*

Definition 22. *A language L is a subset of $\{0,1\}^*$. A* TM *M decides L if M always halts (always halts for any nondeterministic choice, in case of* NTM*), and it accepts any input x if $x \in L$ and rejects x otherwise.*

A complexity class is a set of languages. We first introduce the complexity class with bounded running time or space and the definition of polynomial time (space) TM.

Definition 23. *Let $f(n)$ be a function $\mathbb{N} \to \mathbb{N}$.*

- *A language $L \in$* DTIME*$(f(n))$, if there exists a* DTM *M whose running time is bounded by $O(f(n))$ such that M decides L.*
- *A language $L \in$* NDTIME*$(f(n))$, if there exists a* NTM *M whose running time is bounded by $O(f(n))$ such that M decides L.*
- *A language $L \in$* DSPACE*$(f(n))$, if there a* DTM *M whose space is bounded by $O(f(n))$ such that M decides L.*
- *A language $L \in$* NDSPACE*$(f(n))$, if there exists a* NTM *M whose space is bounded by $O(f(n))$ such that M decides L.*

Definition 24. *A* DTM *(*NTM*) is a polynomial-time (space)* DTM *(*NTM*) if its running time (space) can be bounded by a polynomial function .*

Now we can formally define time/space bounded complexity class P, NP, PSPACE and NPSPACE.

Definition 25. *Complexity class* P *(*NP*) is the set of languages that can be decided by a polynomial-time* DTM *(*NTM*).*

Definition 26. *Complexity class* PSPACE *(*NPSPACE*) is the set of languages that can be decided by a polynomial-space* DTM *(*NTM*).*

Further restricting the amount of available space, we can define the complexity class LOGSPACE.

Definition 27. *Complexity class* LOGSPACE *is the set of languages that can be decided by a* DTM *M whose space is bounded by $O(\log(n))$.*

We have NPSPACE=PSPACE, from Savitch's Theorem [19].

In order to define complete language for certain complexity class, we present the definition of polynomial-time reducible here.

Definition 28. *A language $L \subset \{0,1\}^*$ is polynomial-time reducible to a language $L' \subset \{0,1\}^*$, denoted by $L \leq_p L'$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in L'$.*

We say that L' is NP*-hard if for any language $L \in$ NP, $L \leq_p L'$. We say that L' is* NP*-complete if L' is* NP*-hard and $L' \in$ NP.*

By Cook-Levin Theorem [6], SAT is NP-*complete*.

The definition of PSPACE-*complete* comes naturally.

Definition 29. *We say that L' is* PSPACE*-hard if for any language $L \in$ PSPACE, $L \leq_p L'$. We say that L' is* PSPACE*-complete if L' is* PSPACE*-hard and $L' \in$ PSPACE.*

In addition, define an *oracle machine* as follows:

Definition 30. *A deterministic/nondeterministic oracle machine is a deterministic/nondeterministic Turing machine with oracle access to a language $\mathcal{O}$.*

One can similarly define an oracle machine with two or more oracles. When considering the running time for TM M with oracle access to the language $\mathcal{O}$, it is standard to assume each call of $\mathcal{O}$ requires one unit of time. It is then possible to define P(NP) with oracle access to a language $\mathcal{O}$ or a complexity class C.

Definition 31. *Let $\mathcal{O}$ be a language. Complexity class $\mathsf{P}^{\mathcal{O}}$ ($\mathsf{NP}^{\mathcal{O}}$) is the set of languages such that can be decided by a polynomial-time* DTM *(*NTM*) with oracle access to $\mathcal{O}$.*

Definition 32. *Let $\mathcal{C}$ be a complexity class, define $\mathsf{P}^{\mathcal{C}} = \bigcup_{\mathcal{O} \in C} \mathsf{P}^{\mathcal{O}}$. $\mathsf{NP}^{\mathcal{C}} = \bigcup_{\mathcal{O} \in \mathcal{C}} \mathsf{NP}^{\mathcal{O}}$.*

We also present the definition of polynomial hierarchy (PH). It is a hierarchy of complexity classes that generalize the classes P, NP and coNP to oracle machines.

Definition 33. *Define $\sum_1^{\mathsf{P}} = \mathsf{NP}$, $\sum_{\mathsf{i}}^{\mathsf{P}} = \mathsf{NP}^{\sum_{\mathsf{i}-1}^{\mathsf{P}}}$ for $i > 1$. $\mathsf{PH} = \bigcup_i \sum_{\mathsf{i}}^{\mathsf{P}}$ for $i \in \mathbb{N}$ and $i \geq 1$.*

PH collapses if there exists an i such that $\mathsf{PH} = \sum_{\mathsf{i}}^{\mathsf{P}}$. It is widely believed that PH does not collapse.

We also introduce the problem $\sum_i$SAT which is a $\sum_{\mathsf{i}}^{\mathsf{P}}$-*complete* problem proposed in [14].

Definition 34. *$\sum_i SAT = \exists u_1 \forall u_2 \ldots \mathrm{Q}_i u_i \ \varphi(u_1, u_2, \ldots, u_i) = 1$, where φ is a boolean formula, each u_i is a vector of boolean variables, Q_i is $\forall$ or $\exists$ depending on if i is even or odd.*