國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

存在性二階布林運算式的隨機可驗證系統和其不可估計性

A PCP Protocol for Existential SOQBF and the Inapproximability Result

呂侑承

Yu-Cheng Lu

指導教授: 陳偉松 博士

Advisor: Tony Tan, Ph.D.

中華民國 112 年 7 月

July, 2023

doi:10.6342/NTU202300853

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

存在性二階布林運算式的隨機可驗證系統和其不可估計性

A PCP Protocol for Existential SOQBF and the Inapproximability Result

本論文係呂侑承君（學號 R10922030）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 112 年 7 月 3 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 3 July 2023 have examined a Master's thesis entitled above presented by YU-CHENG LU (student ID: R10922030) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

（指導教授 Advisor）　　江介宏

系主任/所長 Director:

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

致謝

首先，我非常感謝陳偉松教授耐心地指導我研究和寫作。再來，我想感謝我的排球隊隊友、重訓夥伴和我的朋友們對我的支持和陪伴。最後，我想感謝我的家人對我的愛和鼓勵並且總是支持我所做出的決定。

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

Acknowledgements

First, I am grateful to have Tony Tan as my supervisor, who passionately teaches me research methods and writing skills. Second, I would like to thank my volleyball teammates, workout partners, and all my friends for their support and company. Lastly, I must thank my family for their love and encouragement, who always support my decision.

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

摘要

隨機可驗證系統 (probabilistic checkable proof) 是一種證明系統，讓驗證證明的驗證者只需隨機查看部份證明內容即可高機率確認其正確性。事實上，許多被認為困難的問題包含非確定性多項式時間計算 (non-deterministic polynomial-time computation) 和非確定性指數時間計算 (non-deterministic exponential-time computation) 都能使用此種證明系統，並且驗證者只須隨機查看證明中常數個位元的內容即可高機率確認證明的正確性。此結果被稱為隨機可驗證系統理論 (the PCP theorem)，在資訊理論和現實世界中都有許多應用。其中一個應用領域為可驗證計算 (verifiable computation)，此領域的研究者嘗試建立一套機制，使得人們能快速驗證其他人是否正確執行某項電腦計算。過去十年來，研究者提出各種不同的方法來實現可驗證計算，不過他們的方法和討論都集中於（非）確定性多項式時間。近幾年，研究者開始探討超過多項式時間或空間的（非）確定性計算。本論文將嘗試使用隨機可驗證系統建構非確定性指數時間計算的可驗證計算。準確來說，本論文先利用存在性二階布林運算式 (existential second-order boolean formula) 來精簡地表示任何非確定性指數時間計算的計算表，再為存在性二階布林運算式設計一隨機可驗證系統。另外，本論文也證明估計存在性二階布林運算式滿足與否為非確定性指數時間完備問題。

關鍵字：隨機可驗證系統、可驗證計算、非確定性指數時間計算、存在性二階布林運算式

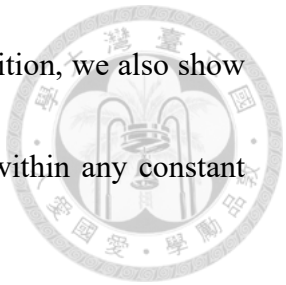
doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

Abstract

Probabilistic checkable proof (PCP) is a type of proof format that could be verified with high probability by only looking at a small portion of the proof. It turns out that problems in NP and even in NEXP have PCP proofs that could be verified in polynomial time by only looking at a constant number of bits in the proof. The result is known as the PCP theorem and has both practical and theoretical significance. One of its practical applications lies in the field of verifiable computation, which aims to provide a scheme for one to efficiently verify if a computation is carried out correctly by others. There are many different approaches in the field while most of the work are dedicated to deterministic/non-deterministic computation in polynomial time. Recently, some researchers investigate the possibility of constructing schemes for deterministic/non-deterministic computation beyond polynomial time and space. In the thesis, we try to construct a scheme for non-deterministic computation in exponential time based on PCP proofs. Specifically, we succinctly encode any non-deterministic exponential computation tableau by a logic

doi:10.6342/NTU202300853



known as ∃SOQBF and construct a PCP proofs for ∃SOQBF. In addition, we also show that it is NEXP-hard to approximate the satisfiability of ∃SOQBF within any constant factor.

Keywords: Probabilistic checkable proof, Verifiable computation, Non-deterministic exponential-time computation, Existential second-order quantified boolean formula

doi:10.6342/NTU202300853

Contents

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

List of Tables

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

Chapter 1 Introduction

In mathematics, one usually proves a theorem by writing down a proof that could be verified by others based on the axioms and the rules of inference. Motivated by automatically theorem proving, the complexity class NP [17, 30] is defined to capture the set of languages whose proofs of membership are efficiently verifiable as follows. For every NP language, there are two entities involved, a prover who generates proofs and an efficient verifier who verifies proofs in polynomial time, such that (1) if the input is in the language then the prover could provide a short proof that passes the verification and (2) conversely, if the input is not in the language then any short proof generated by the prover is rejected by the verifier.

As a generalization, researches study a variant of proof formats that are efficiently verifiable in different proof systems and their corresponding complexity classes [2, 3, 6, 9, 20, 26]. One of the proof formats is known as probabilistic checkable proof (PCP) [2, 20]. In the PCP proof system, there are two entities involved, an oracle who generates proofs and a verifier who verifies proofs. The verifier has random access to the proof generated by the oracle. Namely, whenever it sends an integer i to the oracle, the oracle would return the i-th bit of the proof. A language has a PCP proof if there is a polynomial-time verifier and an oracle such that (1) if the input is in the language then the oracle could generate a proof that passes the verification with high probability (2) conversely, if the input is not

doi:10.6342/NTU202300853

in the language then any proof generated by the oracle is rejected by the verifier with high probability.

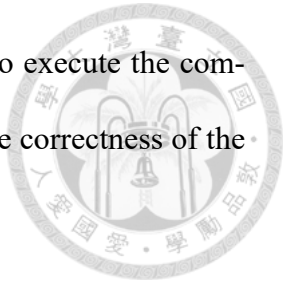
There are two main differences between an NP language and a language with PCP proof. First, the verifier for PCP proof is a probabilistic Turing machine that can verify the proofs with high probability while the verifier for NP do so deterministically. Second, the verifier for PCP proof only needs to read a portion of proof while the verifier for NP reads the entire proof.

As it turns out, PCP proof gives rise to a new characterization of NP language [1, 2]: every NP language has a PCP proof where the verifier could verify the proof by only reading a constant number of bits from the proof generated by the oracle. The result is known as the PCP theorem [1, 2] and has both practical and theoretical significance. In theory, the PCP theorem is a powerful tool to prove hardness of approximation results for optimization problems [19, 44] and it has also initiated the study of locally testable code [24, 25]. In practice, it is the building block for verifiable computation [21] which is used in cloud computing [45] and crypto currency [37].

In the thesis, we are interested in the practical application of PCP proof in verifiable computation. Verifiable computation is a research field that aims to provide solutions to the following question: how to efficiently verify if a computation is carried out correctly by others? There are several desired properties for a verifiable computation scheme. First, the scheme is complete and sound. Namely, if a computation is executed correctly, then one, who executes the computation, could prove to others that it is executed correctly while if a computation is not executed correctly, then no one could prove to others that it is executed correctly. Second, there is little time and space overhead for one to execute

doi:10.6342/NTU202300853

the computation and prove the correctness of the computation than to execute the computation along. Third, it takes less time and space for one to verify the correctness of the computation than to execute the computation.



Verifiable computation is used in cloud computing [45] and crypto currency [37]. In cloud computing, it is crucial for the client who outsources a computation to a server to be able to verify the correctness of the computation since the server is not always trustworthy and might even be malicious [45]. In crypto currency, in order to protect users' privacy while maintaining the decentralization of the currency, users conduct transaction by performing cryptographic computation to hide their data as well as publishing a proof of the computation that enable others to verify the correctness of the transaction [37].

There are many kinds of verifiable computation schemes [10, 11, 13, 16, 18, 22, 32, 35, 40, 41] while most of them are dedicated to deterministic/non-deterministic polynomial time computation. Recently, some researchers investigate the possibility of constructing verifiable computation schemes for deterministic/non-deterministic computation beyond polynomial time and space [7, 27, 43]. In the thesis, we construct a PCP proof for every language in NEXP, which might be useful to construct verifiable computation scheme for non-deterministic exponential-time computation. Specifically, we use the reduction in [15] to succinctly encode non-deterministic exponential-time computation tableaux by a logic known as ∃SOQBF and design PCP proofs for ∃SOQBF. In addition, we also show that it is NEXP-hard to approximate the satisfiability of ∃SOQBF within any constant factor.

There are some advantages of our construction, which are inherited from the reduction in [15]. First, our encoding of computation tableaux is natural in the sense that the

doi:10.6342/NTU202300853

encoding is essentially the Cook-Levin reduction [17, 30] disguised in the form of function certificates. Second, many concrete NEXP-complete problems [34] such as (succinct) 3-SAT, 3-colorability, Hamiltonian cycle, set packing and subset sum could be directly reduced to ∃SOQBF. In comparison with the reduction in [36], our reduction incurs less computation overhead. Thus, our PCP proofs for ∃SOQBF can be transformed as the PCP proofs for those practical problems. Third, it is easy to generalize our encoding for non-deterministic computation with any time complexity and space complexity, which might be useful to construct verifiable scheme for general non-deterministic computation with arbitrary time and space complexity.

doi:10.6342/NTU202300853

Chapter 2 Preliminary

2.1 Existential second-order quantified boolean formula

Existential second-order quantified boolean formula, denoted by ∃SOQBF, is the set of quantified boolean formulas with existential second-order quantifiers. To define ∃SOQBF formulas, we first extend some standard definitions regarding formulas.

Definition 2.1. *Let $\bar{f} = (f_1, \ldots, f_n)$ where each f_i is a function symbol associated with an arity $\mathsf{ar}(f_i)$. Let $\bar{x} = (x_1, \ldots, x_m)$ where each x_i is a boolean variable. Let $\tau = \{f_1, \ldots, f_n, x_1, \ldots, x_m\}$.*

Definition 2.2. *A term over τ is defined as follows.*

- *For every $i \in [m]$, x_i is a term.*
- *For every $i \in [n]$ and $\bar{y} \in \{0, 1, x_1, \ldots, x_m\}^{\mathsf{ar}(f_i)}$, $f_i(\bar{y})$ is a term.*

Definition 2.3. *A boolean formula over τ is a well-formed expression built from terms over τ using operations $\wedge, \vee, \neg$.*

Definition 2.4. *A quantified boolean formula, denoted by* QBF *formula, over τ is a formula of the form $Q_1 x_1 \ldots Q_m x_m \psi(\bar{x}, \bar{f})$ where each $Q_i \in \{\exists, \forall\}$ and $\psi(\bar{x}, \bar{f})$ is a boolean formula over τ.*

doi:10.6342/NTU202300853

Definition 2.5. *An* ∃SOQBF *formula* $\Psi(\bar{x}, \bar{f})$ *over* τ *is of the form*

$$\Psi(\bar{x}, \bar{f}) = \exists f_1 \dots \exists f_n Q_1 x_1 \dots Q_m x_m \ \psi(\bar{x}, \bar{f})$$

where $Q_1 x_1 \dots Q_m x_m \psi(\bar{x}, \bar{f})$ *is a* QBF *formula over* τ.

A formula $\Psi(\bar{x}, \bar{f}) = \exists f_1 \dots \exists f_n Q_1 x_1 \dots Q_m x_m \ \psi(\bar{x}, \bar{f})$ *is satisfiable if there is an interpretation* $\bar{F} = (F_1, \dots, F_n)$ *where each* F_i *is a boolean function with arity* $\mathsf{ar}(f_i)$ *such that* $Q_1 x_1 \dots Q_m x_m \ \psi(\bar{x}, \bar{F})$ *is a true* QBF.

Given a formula $\Psi(\bar{x}, \bar{f}) = \exists f_1 \dots \exists f_n Q_1 x_1 \dots Q_m x_m \ \psi(\bar{x}, \bar{f})$ with $Q_i = \exists$ for some i, we could transform it into an equisatisfiable formula where every quantifier of boolean variable is $\forall$. Let $x_{j_1}, \dots, x_{j_s}$ be boolean variables whose quantifier is $\exists$ and let $x_{k_1}, \dots, x_{k_t}$ be boolean variables whose quantifier is $\forall$. For each j_i with $Q_{j_i} = \exists$, introduce a new function symbol f'_{j_i} of arity $j_i - i$ and replace any occurrence of x_{j_i} in formula $\psi(\bar{x}, \bar{f})$ by $f'_{j_i}(\bar{y})$ where $\bar{y} = \{x_1, \dots, x_{j_i}\} \setminus \{x_{j_1}, \dots, x_{j_i}\}$. Denote by ψ' the resulted formula. Then, define Ψ' by

$$\Psi(x_{k_1}, \dots, x_{k_t}, f'_{j_1}, \dots, f'_{j_s}, \bar{f}) \stackrel{\mathsf{def}}{=} \exists f_1 \dots \exists f_n \exists f'_{j_1} \dots \exists f'_{j_s} \forall x_{k_1} \dots \forall x_{k_t} \ \psi'$$

Clearly, Ψ' is equisatisfiable with Ψ. Thus, we may assume that every ∃SOQBF formula is of the form $\exists f_1 \dots \exists f_n \forall x_1 \dots \forall x_m \ \psi(\bar{x}, \bar{f})$.

Given a formula $\Psi(\bar{x}, \bar{f})$, we could transform it into an equisatisfiable formula where the arity of each function symbol is m by introducing new function symbols $g_1, \dots, g_n$ and replacing each occurrence of $f_i(\bar{y})$ by $g_i(\bar{y}, \bar{0})$ where $\bar{0} = (0, \dots, 0)$ is a string of length $m - \mathsf{ar}(f_i)$. Thus, we may assume that the function symbol of every ∃SOQBF formula over τ are of the same arity m.

doi:10.6342/NTU202300853

As pointed out in [15], ∃SOQBF is an equivalent formalism of dependency quantified boolean formula, denoted by DQBF, which is a well-studied logic motivated by application in verification and synthesis of hardware and software design [8, 12, 14, 23, 28, 29, 38]. In particular, there is a linear-time reduction between them which preserves satisfiability. Since it is NEXP-complete to decide if a DQBF formula is satisfiable [36], the same holds for ∃SOQBF.

Definition 2.6. SAT(∃SOQBF) *is the problem of, given an* ∃SOQBF *formula, to decide if it is satisfiable.*

Proposition 2.7 ([36])**.** SAT(∃SOQBF) *is* NEXP*-complete.*

We also define the optimization problem and promise problem of ∃SOQBF formulas in a similar fashion as MAX NP and MAX-3SAT were defined [33].

Definition 2.8. *For every formula* $\Psi(\bar{x}, \bar{f}) = \exists f_1 \dots \exists f_n \forall x_1 \dots \forall x_m \, \psi(\bar{x}, \bar{f})$ *and interpretation* $\bar{F} = (F_1, \dots, F_n)$*, define* $\sharp\text{SAT}(\Psi, \bar{F})$ *as*

$$\sharp\text{SAT}(\Psi, \bar{F}) = |\{\bar{a} \in \{0,1\}^m : \psi(\bar{a}, \bar{F}) = 1\}|$$

For $\delta > 0$*, an* ∃SOQBF *formula* Ψ *is* δ*-satisfiable if*

$$\max_{\bar{F} \text{ is an interpretation}} \sharp\text{SAT}(\Psi, \bar{F}) \geq \delta \cdot 2^m$$

Definition 2.9. MAX-SAT(∃SOQBF) *is the problem of, given an* ∃SOQBF *formula* Ψ*, to find an interpretation* $\bar{F}$ *that maximizes* $\sharp\text{SAT}(\Psi, \bar{F})$.

Definition 2.10. *For* $\delta \in (0, 1)$*,* δGAP-SAT(∃SOQBF) *is the problem of, given an* ∃SOQBF *formula that is either satisfiable or* δ*-satisfiable, to decide if it is satisfiable.*

doi:10.6342/NTU202300853

2.2 Polynomials with function symbols

In the construction of the PCP proof for an ∃SOQBF formula and its interpretation, we arithmetize the formula and the interpretation as polynomials. To do so, we would need to extend the definitions regarding polynomials and arithmetization. The standard and extended definitions are as follows.

Definition 2.11. *A polynomial $p(\bar{x})$ is a well-formed expression built from variables in $\bar{x}$ and integers using operations $+, -, \times$.*

Definition 2.12. *A polynomial $p(\bar{x})$ is an arithmetization of a boolean function $F : \{0,1\}^m \to \{0,1\}$ if for every $\bar{a} \in \{0,1\}^m$, $p(\bar{a}) = F(\bar{a})$.*

Definition 2.13. *A polynomial $p(\bar{x}, \bar{f})$ over τ is a well-formed expression built from terms over τ and integers using operations $+, -, \times$.*

Definition 2.14. *A polynomial $p(\bar{x}, \bar{f})$ over τ is an arithmetization of a boolean formula $\psi(\bar{x}, \bar{f})$ over τ if for every $\bar{a} \in \{0,1\}^m$ and boolean functions $\bar{F} = (F_1, \ldots, F_n)$, $p(\bar{a}, \bar{F}) = \psi(\bar{a}, \bar{F})$.*

Finally, we define some terminology that is used in the design and analysis of the protocol that checks the correctness of the PCP proofs for ∃SOQBF formulas.

Definition 2.15. *For every polynomial $p(\bar{x})$, let $\deg(p)$ denote the degree of p and for each $i \in [m]$, let $\deg_i(p)$ denote the degree of p in variable x_i.*

Definition 2.16. *A polynomial $p(\bar{x})$ is multilinear if for each $i \in [m]$, $\deg_i(p) \leq 1$.*

doi:10.6342/NTU202300853

2.3 Probabilistically checkable proof

Probabilistically checkable proof, denoted by PCP proof, is a type of membership proof of language that could be verified by randomly reading a small portion of the proof. A PCP proof system involves two entities, a verifier and an oracle. In the system, the oracle is a function that stores a proof while the verifier is a randomized algorithm with access to the oracle that verifies the proof. The definitions of the entities are as follows.

Definition 2.17. *An oracle is a function* $\pi : \{0,1\}^* \to \{0,1\}$.

An oracle π stores a proof in an obvious way that $\pi(i)$ is the i-th bit of the proof.

Definition 2.18. *A probabilistic oracle Turing machine is a probabilistic Turing machine that has a query state and two extra tapes, a query tape and an answer tape. To run such a machine, an oracle* π *is required and whenever the machine enters the query state and the query tape contains a string* i*, the oracle writes* $\pi(i)$ *in the answer tape accordingly.*

Definition 2.19. *A PCP verifier is a polynomial-time probabilistic oracle Turing machine. Denote by* $V^\pi(x, r)$ *the output of a verifier* V *with access to an oracle* π *given input* x *and random bits* r.

Definition 2.20. *For functions* $r, q : \mathbb{N} \to \mathbb{N}$*, an* $(r(n), q(n))$*-PCP verifier* V *is a verifier which on input of length* n*, uses at most* $r(n)$ *random bits and queries at most* $q(n)$ *bits from the oracle.*

The definition of a language having a proof system is as follows.

Definition 2.21. *For functions* $r, q : \mathbb{N} \to \mathbb{N}$*, a language* L *has an* $(r(n), q(n))$*-PCP proof, denoted by* $L \in \text{PCP}[r(n), q(n)]$*, if there is an* $(r(n), q(n))$*-PCP verifier* V *such that for every* $x \in \{0,1\}^*$ *the following holds.*

doi:10.6342/NTU202300853

- *If $x \in L$, then there is an oracle π such that* $\mathbf{Pr}_r[\, V^\pi(x, r) = \mathsf{ACCEPT} \,] = 1$.
- *If $x \notin L$, then for every oracle π,* $\mathbf{Pr}_r[\, V^\pi(x, r) = \mathsf{ACCEPT} \,] \leq \frac{1}{2}$.

The definition introduces a new hierarchy of complexity classes, which classifies languages based on how efficiently their membership proof could be probabilistically checked by a verifier. As it turns out, PCP proof gives rise to a new characterization of NP, a class of languages whose membership proof could be verified by a polynomial-time Turing machine.

Theorem 2.22 (PCP theorem [1, 2])**.** $\mathrm{NP} = \mathrm{PCP}[O(\log n), O(1)]$.

Corollary 2.23 ([1, 2])**.** $\mathrm{NEXP} = \mathrm{PCP}[\mathit{poly}(n), O(1)]$.

doi:10.6342/NTU202300853

Chapter 3 Encoding of NEXP computation by ∃SOQBF

In this chapter, we show the way to encode any NEXP computation by ∃SOQBF by proving Proposition 2.7 by modifying the proof in [15]. We also use ∃SOQBF to encode the computation of probabilistic oracle machines to prove that δGAP-SAT(∃SOQBF) is NEXP-hard for every $\delta \in (0, 1)$.

Proposition 3.1 ([15])**.** SAT(∃SOQBF) *is* NEXP-*complete.*

Proof. Let L be an NEXP language decided by an 1-tape NTM $\mathcal{M}$ in time $2^{t(n)}$ for some polynomial $t(n)$. Let Q and Γ be the set of states and the tape alphabet of $\mathcal{M}$ respectively. For simplicity, we assume that $\mathcal{M}$ only accepts exactly at step $2^{t(n)}$ while the head is in the leftmost cell. We also assume that when $\mathcal{M}$ makes a non-deterministic move, the head stays still.

Let $\Delta = \Gamma \cup (Q \times \Gamma)$. An accepting run of $\mathcal{M}$ on w is represented by a function $g : [2^{t(n)}] \times [2^{t(n)}] \to \Delta$, where $g(i, j)$ is the symbol in cell i in time j in the run. When $g(i, j) \in Q \times \Gamma$, it indicates the position of the head is in cell i and the state of $\mathcal{M}$.

Let the input word w be $b_0 b_1 \cdots b_{n-1}$. For a function $g : [2^{t(n)}] \times [2^{t(n)}] \to \Delta$ to represent a correct accepting run of $\mathcal{M}$ on w, the following must hold for every $i, j, i', j' \in$

doi:10.6342/NTU202300853

$[2^{t(n)}]$.

(a) If $i = j = 0$, then $g(i, j) = (q_0, b_0)$.

(b) If $1 \leq i \leq n - 1$ and $j = 0$, then $g(i, j) = b_i$.

(c) If $n \leq i$ and $j = 0$, $g(i, 0)$ is the blank symbol.

(d) If $j = j'$, then at most one of $g(i, j)$ and $g(i', j')$ is an element of $Q \times \Gamma$.

(e) If $i = 0$ and $j = 2^{t(n)} - 1$, then $g(i, j) = (q_{acc}, \sigma)$ for some tape symbol σ where q_{acc} is the accepting state of $\mathcal{T}$.

(f) If $|i - i'| \leq 1$ and $|j - j'| \leq 1$ (modulo $2^{t(n)}$), then $g(i, j)$ and $g(i', j')$ must obey the transitions in $\mathcal{T}$.

For example, if there is a transition $(s, 0) \to (s', 1, \text{stay})$ and $(s, 0) \to (s'', 0, \text{stay})$, we have the following condition.

- If $i = i'$ and $j' = j + 1$ and $g(i, j) = (s, 0)$, then $g(i, j') = (s', 1)$ or $(s'', 0)$.

Similar condition can be defined for each transition.

Obviously, $[2^{t(n)}]$ can be encoded with $\{0, 1\}^{t(n)}$ and Δ with $\{0, 1\}^{\ell}$, where $\ell = \log |\Delta|$. By abusing the notation, let $i = (i_1, \ldots, i_{t(n)}), (j_1, \ldots, j_{t(n)}), i' = (i'_1, \ldots, i'_{t(n)}), j' = (j'_1, \ldots, j'_{t(n)})$ be strings of boolean variables and $g = (g_1, \ldots, g_\ell)$ be function symbols with arity $2t(n)$. Let $\tau' = \{i, j, i', j', g\}$.

It is routine to design an algorithm (with access to the transitions in $\mathcal{M}$) that on input w, constructs a boolean formula ψ over τ' that verifies whether all properties (a)–(f) hold for $i, j, g(i, j), i', j', g(i', j')$. That is, $\psi(i, j, i', j', g) = 1$ iff $(i, j, g(i, j), i', j', g(i', j'))$ satisfies (a)–(f). In other words, $\psi(i, j, i', j', g) = 1$ for every possible i, j, i', j' iff g

doi:10.6342/NTU202300853

represents a correct accepting run of M on w. Thus, the resulting ∃SOQBF formula $\Psi(i, j, i', j', g)$ is as follows

$$\Psi(i, j, i', j', g) \stackrel{\text{def}}{=} \exists g \forall i \forall j \forall i' \forall j' \psi(i, j, i', j', g)$$

where $\exists g$ stands for $\exists g_1 \ldots \exists g_\ell$ and the similar holds for $\forall i, \forall j, \forall i', \forall j'$. □

Remark 3.2. *Note that since $\Delta = \Gamma \cup (Q \times \Gamma)$, $\ell = \log |\Delta| = O(1)$. It follows that there is only constant number of function symbols $g_1, \ldots, g_\ell$ in the resulting* ∃SOQBF *formula. Also, by introducing some dummy boolean variable, we could rewrite formula Ψ such that the number of occurrences of the function symbols in the formula becomes a constant. (Specifically, we need to construct a circuit b and use it in checking property (b) such that for every input i, $b(i)$ outputs b_i, the i-th bit of the input word w.)*

Remark 3.3. *Note that we could use a function $g : [T(n)] \times [S(n)] \to \Delta$ to represent an accepting run of an non-deterministic Turing machine in $T(n)$ time and with $S(n)$ space on an input of length n. Thus, it is easy to modify the proof for general non-deterministic computation.*

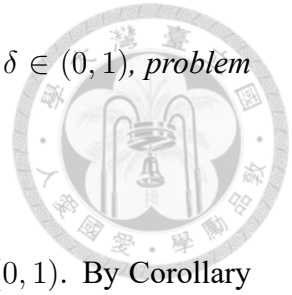
To show that δGAP-SAT(∃SOQBF) is NEXP-hard for every $\delta \in (0, 1)$, we need the following corollary of the PCP theorem 2.22.

Corollary 3.4. *For every language $L \in$* NEXP *and $\delta \in (0, 1)$, there is a $(\mathsf{poly}(n), O(1))$-PCP verifier for L such that the following holds.*

- *If $x \in L$, then there is an oracle π such that* $\mathbf{Pr}_r[\, V^\pi(x, r) = \mathsf{ACCEPT} \,] = 1$.
- *If $x \notin L$, then for every oracle π,* $\mathbf{Pr}_r[\, V^\pi(x, r) = \mathsf{ACCEPT} \,] \leq \delta$.

doi:10.6342/NTU202300853

Theorem 3.5 (The inapproximability of SAT(∃SOQBF))**.** *For every* $\delta \in (0,1)$*, problem* δGAP-SAT(∃SOQBF) *is* NEXP*-hard.*



Proof. Let L be a language in NEXP and let δ be a real number in $(0,1)$. By Corollary 3.4, there is a $(\mathsf{poly}(n), O(1))$-PCP verifier V for L such that for every $x \in \{0,1\}^*$ the following holds.

- If $x \in L$, then there is an oracle π such that $\mathbf{Pr}_r[\, V^\pi(x,r) = \mathsf{ACCEPT}\,] = 1$.
- If $x \notin L$, then for every oracle π, $\mathbf{Pr}_r[\, V^\pi(x,r) = \mathsf{ACCEPT}\,] \leq \delta$

Let $x \in \{0,1\}^n$ be an input. Let $p(n)$ be the number of random bits that the verifier uses and let $q(n)$ be the maximum length of a possible query that the verifier sends to the oracle. Let $t(n)$ be the running time of V. Let g be a function symbol with arity $q(n)$ and let $\bar{h} = (h_1, \ldots, h_{t^2(n)})$ be function symbols with arity $q(n)$.

In the following, we encode the computation tableau of verifier V on input x and random string r as a boolean formula $\psi(r, g, \bar{h})$ with function symbols g and $\bar{h}$ such that for every random string r and oracle π, $V^\pi(x,r) = 1$ if and only if there are functions $\bar{H} = (H_1, \ldots, H_{t^2(n)})$ such that $\psi(r, \pi, \bar{H}) = 1$. In the formula ψ, each boolean variable and function symbol is associated with a cell in the computation tableau. Specifically, each variable in r represents a random bit, each occurrence of g represents an oracle answer, and each $h_i(r)$ represents a cells other than a random bit or an oracle answer. Similar to the Cook-Levin theorem [17, 30], we could construct the formula $\psi(r, g, \bar{h})$ such that for every random string r, function π and functions $\bar{H} = (H_1, \ldots, H_{t^2(n)})$, $\psi(r, \pi, \bar{H})$ evaluates to 1 if and only if $(r, \pi, \bar{H})$ represents an accepting computation of V on input x and random string r.

doi:10.6342/NTU202300853

Define the ∃SOQBF formula $\Psi(r, g, \bar{h})$ as follows.

$$\Psi(r, g, \bar{h}) \stackrel{\text{def}}{=} \exists g \exists \bar{h} \forall r \psi(r, g, \bar{h})$$

where $\exists \bar{h}$ stands for $\exists h_1 \ldots \exists h_{t^2(n)}$ and $\forall r$ stands for $\forall r_1 \ldots \forall r_{p(n)}$. If $x \in L$, then there exists an oracle π such that $V^\pi(x, r) = 1$ for every r. It follows that there are functions $\bar{H} = (H_1, \ldots, H_{t^2(n)})$ such that $\psi(r, \pi, \bar{H}) = 1$ and thus Ψ is satisfiable. If $x \notin L$, then for every oracle π, $\mathbf{Pr}_r[\, V^\pi(x, r) = 0 \,] > 1 - \delta$. Thus, if $V^\pi(x, r) = 0$ for some random string r and some oracle π, then $(r, \pi, \bar{H})$ does not represent an accepting computation of V on input x and random string r for any functions $\bar{H}$, which implies that $\psi(r, \pi, \bar{H}) = 0$ for any functions $\bar{H}$. Therefore, for every oracle π and functions $\bar{H}$, $\mathbf{Pr}_r[\, \psi(r, \pi, \bar{H}) = 0 \,] \geq \mathbf{Pr}_r[\, V^\pi(x, r) = 0 \,] > 1 - \delta$ and it follows that Ψ is not δ-satisfiable. □

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

Chapter 4 The PCP proof for SAT(∃SOQBF)

In the following two chapters, we design a PCP proof system for SAT(∃SOQBF). In this chapter, we show that for every satisfiable ∃SOQBF formula, we can construct a PCP proof based on a satisfying interpretation while in Chapter 5, we design a PCP verifier such that given an ∃SOQBF formula, the following holds.

- If the formula is satisfiable, then the verifier always accepts the oracle that stores the PCP proof constructed as in this chapter.
- Otherwise, the verifier rejects any oracle with high probability.

At the end of Chapter 5, we prove our main result.

Theorem 4.1. SAT(∃SOQBF) $\in$ PCP$[n\log(n), n^4\log(n)]$.

We define some notation for the following two chapters.

Notation 4.2. *For every* ∃SOQBF *formula* $\Psi = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m\ \psi(\bar{x}, \bar{f})$*, let* k *be the length of* ψ *and let* ℓ *be the number of occurrences of function symbols in* ψ*. We fix a set* $\mathcal{I}$ *of integers* $\{0, 1, \ldots, N-1\}$ *with* $N = 7mk$*. We also fix a finite field with* $mN^{m+2} < |\mathbb{K}| < 2mN^{m+2}$ *and a subset* $H \subset \mathbb{K}$ *with* $|H| = 200m\ell$*.*

doi:10.6342/NTU202300853

Recall that in this chapter, we would show that for every satisfiable ∃SOQBF formula, we can construct a PCP proof based on a satisfying interpretation. Let $\Psi(\bar{x}, \bar{f}) = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m \ \psi(\bar{x}, \bar{f})$ be an ∃SOQBF formula and let $\bar{F} = (F_1, \ldots, F_n)$ be a satisfying interpretation for $\Psi(\bar{x}, \bar{f})$. In Section 4.1, we arithmetize each boolean function $F_i(\bar{x})$ as a multilinear polynomial $\mathcal{F}_i(\bar{x})$. In Section 4.2, we arithmetize formula $\Psi(\bar{x}, \bar{f})$ as a set of polynomials by using the idea of arithmetization of QBF formula [42]. In Section 4.3, we construct a string for each polynomial obtained in Section 4.1 and Section 4.2 and the PCP proof based on $\bar{F}$ is simply the concatenation of these strings.

4.1 Arithmetization of interpretations

In this section, we arithmetize interpretations $\bar{F}$ and show properties that will be used to check the arithmetization in Chapter 5. The following proposition arithmetizes each boolean function F_i as a multilinear polynomial $\mathcal{F}_i$.

Proposition 4.3. *For every boolean function $F : \{0,1\}^m \to \{0,1\}$, there is a multilinear polynomial $\mathcal{F}(\bar{x})$ which is an arithmetization of F.*

Proof. Define $\mathcal{F}(\bar{x}) : \mathbb{Z}^m \to \mathbb{Z}$ as follows.

$$\mathcal{F}(\bar{x}) \stackrel{\text{def}}{=} \sum_{\bar{a} \in \{0,1\}^m} F(\bar{a}) \cdot eq(\bar{a}, \bar{x})$$

where $eq(\bar{x}, \bar{y}) = \prod_{i \in [m]} (1 - x_i - y_i + 2x_i y_i)$. Clearly, $\mathcal{F}$ is multilinear. Note that for every $\bar{a}, \bar{b} \in \{0,1\}^m$, if $\bar{a} = \bar{b}$, then $eq(\bar{a}, \bar{b}) = 1$ and if otherwise, then $eq(\bar{a}, \bar{b}) = 0$. Thus, $\mathcal{F}(\bar{a}) = F(\bar{a})$ for every $\bar{a} \in \{0,1\}^m$. □

doi:10.6342/NTU202300853

4.2 Arithmetization of ∃SOQBF formulas

In this section, we arithmetize ∃SOQBF formula $\Psi(\bar{x}, \bar{f})$ along with interpretation $\bar{F}$ and show properties that will be used to check the arithmetization in Chapter 5.

The first step to arithmetize formula $\Psi(\bar{x}, \bar{f})$ is to arithmetize boolean formula $\psi(\bar{x}, \bar{f})$ as a polynomial $p(\bar{x}, \bar{f})$.

Proposition 4.4. *Given a boolean formula $\psi(\bar{x}, \bar{f})$ over τ, we can construct in linear time a polynomial $p(\bar{x}, \bar{f})$ over τ which is an arithmetization of $\psi(\bar{x}, \bar{f})$.*

Proof. Let ψ be a boolean formula. Replace any occurrence of $\neg y$ by $(1 - y)$, $y \wedge z$ by $y \times z$, and $y \vee z$ by $1 - (1 - y)(1 - z)$. Then replace any occurrence of y^k for any $k \in \mathbb{N}$ by y. Clearly, the resulted polynomial is an arithmetization of ψ. □

Notation 4.5. *For every boolean formula $\psi(\bar{x}, \bar{f})$, let $p^{\psi}(\bar{x}, \bar{f})$ denote the arithmetization of $\psi(\bar{x}, \bar{f})$ that one would obtain by using the procedure described in Proposition 4.4.*

The following definition is the second step to arithmetize $\Psi(\bar{x}, \bar{f})$, which uses the idea of arithmetization of QBF formula [42].

Definition 4.6. *For every* ∃SOQBF *formula $\Psi(\bar{x}, \bar{f}) = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m \, \psi(\bar{x}, \bar{f})$, define polynomial $p_m^{\Psi}(\bar{x}, \bar{f})$ by*

$$p_m^{\Psi}(\bar{x}, \bar{f}) \stackrel{def}{=} (p^{\psi}(\bar{x}, \bar{\mathcal{F}}) - 1)^2 + \sum_{i \in [n]} (f_i(\bar{x})(f_i(\bar{x}) - 1))^2$$

and for each $i \in [m-1]$, define polynomial $p_i^{\Psi}(x_1, \ldots, x_i, \bar{f})$ as follows.

$$p_i^{\Psi}(x_1, \ldots, x_i, \bar{f}) \stackrel{def}{=} \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_m \in \{0,1\}} p_m^{\Psi}(\bar{x}, \bar{f})$$

doi:10.6342/NTU202300853

The following properties would be used to check the arithmetization in Chapter 5.

Proposition 4.7. *For every* ∃SOQBF *formula* $\Psi(\bar{x}, \bar{f}) = \exists f_1 \dots \exists f_n \forall x_1 \dots \forall x_m \psi(\bar{x}, \bar{f})$, *multilinear polynomials* $\bar{\mathcal{F}} = (\mathcal{F}_1, \dots, \mathcal{F}_n)$ *and* $i \in [m]$,

$$\deg_i(p_i^{\Psi}(\bar{x}, \bar{\mathcal{F}})) \leq 2k = 2|\psi|$$

Proof. It follows immediately from the definition. □

Proposition 4.8. *For every* ∃SOQBF *formula* $\Psi(\bar{x}, \bar{f})$, *it is satisfiable if and only if there exist multilinear polynomials* $\bar{\mathcal{F}} = (\mathcal{F}_1, \dots, \mathcal{F}_n)$ *such that*

$$\sum_{\bar{a} \in \{0,1\}^m} p_m^{\Psi}(\bar{a}, \bar{\mathcal{F}}) = 0 \tag{4.1}$$

Proof. If $\Psi(\bar{x}, \bar{f})$ is satisfiable, then there exists an interpretation $\bar{F} = (F_1, \dots, F_n)$ such that $\forall x_1 \dots \forall x_m \ \psi(\bar{x}, \bar{F})$ is a true QBF formula. Since $p^{\psi}(\bar{x}, \bar{f})$ is an arithmetization of $\psi(\bar{x}, \bar{f})$, it follows that for every $\bar{a} \in \{0,1\}^m$, $p^{\psi}(\bar{a}, \bar{F}) = 1$. By Proposition 4.3, there exist multilinear polynomials $\bar{\mathcal{F}} = (\mathcal{F}_1, \dots, \mathcal{F}_n)$ such that each $\mathcal{F}_i$ is an arithmetization of boolean function F_i. It follows that for every $\bar{a} \in \{0,1\}^m$, $p^{\psi}(\bar{a}, \bar{\mathcal{F}}) = 1$ and $\mathcal{F}_i(\bar{a})(\mathcal{F}_i(\bar{a}) - 1) = 0$, which implies that $\sum p_m^{\Psi}(\bar{a}, \bar{\mathcal{F}}) = 0$.

If there exists multilinear polynomials $\bar{\mathcal{F}} = (\mathcal{F}_1, \dots, \mathcal{F}_n)$ such that $\sum p_m^{\Psi}(\bar{a}, \bar{\mathcal{F}}) = 0$, then it implies that for every $\bar{a} \in \{0,1\}^m$ and $i \in [n]$, $p^{\psi}(\bar{a}, \bar{\mathcal{F}}) = 1$ and $\mathcal{F}_i(\bar{a})(\mathcal{F}_i(\bar{a}) - 1) = 0$. Since $\mathcal{F}_i(\bar{a}) \in \{0,1\}$ for each $i \in [n]$ and $\bar{a} \in \{0,1\}^m$, each $\mathcal{F}_i$ is an arithmetization of a boolean function F_i. Let $\bar{F} = (F_1, \dots, F_n)$ be the boolean functions where $\mathcal{F}_i$ is an arithmetization of F_i. Since each $\mathcal{F}_i$ is an arithmetization of F_i, it follows that for every $\bar{a} \in \{0,1\}^m$, $p^{\psi}(\bar{a}, \bar{F}) = p^{\psi}(\bar{a}, \bar{\mathcal{F}}) = 1$. Since $p^{\psi}(\bar{x}, \bar{f})$ is an arithmetization of ψ, $\forall x_1 \dots \forall x_m \, \psi(\bar{x}, \bar{F})$ is a true QBF formula and $\Psi(\bar{x}, \bar{f}) = \exists f_1 \dots \exists f_n \forall x_1 \dots \forall x_m \, \psi(\bar{x}, \bar{f})$

doi:10.6342/NTU202300853

is a true ∃SOQBF formula. □

4.3 The PCP proof for SAT(∃SOQBF)

In this section, we construct the PCP proof that an ∃SOQBF formula is satisfied by an interpretation.

Let $\Psi(\bar{x}, \bar{f}) = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m \, \psi(\bar{x}, \bar{f})$ be an ∃SOQBF formula and let $\bar{F} = (F_1, \ldots, F_n)$ be a satisfying interpretation for $\Psi(\bar{x}, \bar{f})$. Let $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ be multilinear polynomials where each $\mathcal{F}_i$ is the arithmetization of F_i obtained as defined in Proposition 4.3. Let $p_1^\Psi, \ldots, p_m^\Psi$ be the polynomials defined in Definition 4.6 and for each $i \in [m]$, let $p_i(x_1, \ldots, x_i) = p_i^\Psi(x_1, \ldots, x_i, \bar{\mathcal{F}})$.

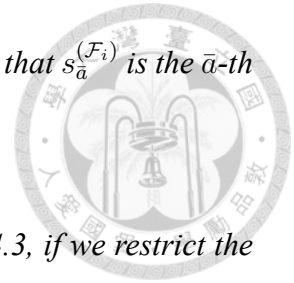
Intuitively, in the PCP proof system for SAT(∃SOQBF), the verifier would query for the values of $\mathcal{F}_i$ and the coefficients of p_i. Since it only has polynomial time to verify the proof, it could not query them arbitrarily or else it might not even have time to read the answer from the oracle. Thus, it queries them only in some restricted domains. For each $\mathcal{F}_i$ and p_i, we construct a string based on the polynomial with a restricted domain where each substring corresponds to a possible query of the verifier. The PCP proof is the concatenation of the strings and the verifier could make queries to the oracle to get corresponding substrings in the concatenation. The details of the verifier are in Chapter 5.

For each $\mathcal{F}_i$, we restrict its domain to $\mathcal{I}^m$. Recall that $\mathcal{I} = \{0, 1, \ldots, N-1\}$ and $N = 7mk$ where $k = |\psi|$.

Notation 4.9. *For each $i \in [n]$, we denote $s^{(\mathcal{F}_i)}$ to be the concatenation of strings $\{s_{\bar{a}}^{(\mathcal{F}_i)}$:*

doi:10.6342/NTU202300853

$\bar{a} \in \mathcal{I}^m\}$ *where each* $s_{\bar{a}}^{(\mathcal{F}_i)}$ *is the binary encoding of* $\mathcal{F}_i(\bar{a})$ *and we say that* $s_{\bar{a}}^{(\mathcal{F}_i)}$ *is the* $\bar{a}$*-th entry of* $s^{(\mathcal{F}_i)}$.



Remark 4.10. *Since each* $\mathcal{F}_i$ *is obtained as defined in Proposition 4.3, if we restrict the domain of* $\mathcal{F}_i$ *to* $\mathcal{I}^m$*, then its codomain is restricted to* $\{0, \pm 1, \ldots, \pm 2^m N^m\}$. *Thus, for each* $i \in [n]$ *and* $\bar{a} \in \mathcal{I}^m$, $s_{\bar{a}}^{(\mathcal{F}_i)}$ *has length* $O(m \log(N))$ *and* $s^{(\mathcal{F}_i)}$ *has length* $O(N^m \cdot m \log(N))$.

Proposition 4.11. *Given a function* $F : \{0,1\}^m \to \{0,1\}$ *and an arbitrary* $\bar{a} \in \mathcal{I}^m$*, it takes* $O(2^m m^2 \log N \log m \log\log m)$ *time to compute the value of* $\mathcal{F}(\bar{a})$ *where* $\mathcal{F} : \mathbb{Z}^m \to \mathbb{Z}$ *is the multilinear function of* F *in Proposition 4.3.*

Proof. By the definition in Proposition 4.3, one perform addition 2^m times and multiplication $m2^m$ times. Since the domain of $\mathcal{F}$ is restricted to $\mathcal{I}^m$, the intermediate values during the addition and multiplication computation are bounded by $\pm 2^m N^m$ and $\pm N^m$, respectively. Thus, it takes $\log(2^m N^m) = O(m \log N)$ time to do each addition and $O(m \log N \log m \log\log m)$ time to do each multiplication by Schönhage-Strassen algorithm [39]. The total time of the computation is $O(2^m m^2 \log N \log m \log\log m)$. □

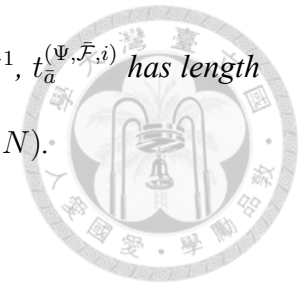
For each $p_i(x_1, \ldots, x_{i-1})$, the verifier would query for the coefficients of $p_i(\bar{a}, x_i)$ for some $\bar{a} \in \mathcal{I}^{i-1}$. To do so, we restrict the domain of each p_i to $\mathcal{I}^i$.

Notation 4.12. *For each* $i \in [m]$*, we denote* $t^{(\Psi,\bar{\mathcal{F}},i)}$ *to be the concatenation of strings* $\{t_{\bar{a}}^{(\Psi,\bar{\mathcal{F}},i)} : \bar{a} \in \mathcal{I}^{i-1}\}$ *where each* $t_{\bar{a}}^{(\Psi,\bar{\mathcal{F}},i)}$ *is the binary encoding of the coefficients of polynomial* $p_i(\bar{a}, x_i)$ *and we say that* $t_{\bar{a}}^{(\Psi,\bar{\mathcal{F}},i)}$ *is the* $\bar{a}$*-th entry of* $t^{(\Psi,\bar{\mathcal{F}},i)}$.

Remark 4.13. *Since each* $\mathcal{F}_i$ *is obtained as defined in Proposition 4.3, for every* $\bar{a} \in \mathcal{I}^{i-1}$*, any coefficient of* $p_i(\bar{a}, x_i)$ *is restricted to* $\{0, \pm 1, \ldots, \pm 2^{2mk+m} N^{2mk}\}$. *Also note that by*

doi:10.6342/NTU202300853

Proposition 4.7, $\deg_i(p_i) \leq 2k$. *Thus, for each* $i \in [m]$ *and* $\bar{a} \in \mathcal{I}^{i-1}$, $t_{\bar{a}}^{(\Psi,\bar{\mathcal{F}},i)}$ *has length* $4mk^2 + 4mk^2 \log N$ *and* $t^{(\Psi,\bar{\mathcal{F}},i)}$ *has length* $N^{i-1}(4mk^2 + 4mk^2 \log N)$.



Similarly, we could prove the following proposition.

Proposition 4.14. *Given an* ∃SOQBF *formula* $\Psi(\bar{x}, \bar{f})$*, an interpretation* $F_1, \ldots, F_n$ *and an arbitrary* $\bar{a} \in \mathcal{I}^m$*, it takes* $O(4^m m^2 k^2 \log^2 N \log\log N)$ *time to compute the coefficients of* $t_{\bar{a}}^{(\Psi,\bar{\mathcal{F}},i)}$ *where* $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ *and each* $\mathcal{F}_i : \mathbb{Z}^m \to \mathbb{Z}$ *is the multilinear function of* F_i *in Proposition 4.3.*

Definition 4.15. *For every* ∃SOQBF *formula* $\Psi(\bar{x}, \bar{f})$ *and interpretation* $\bar{F} = (F_1, \ldots, F_n)$*, define the PCP proof that formula* Ψ *is satisfied by interpretation* $\bar{F}$ *as the concatenation of strings* $\{s^{(\mathcal{F}_i)} : i \in [n]\}$, $\{t^{(\Psi,\bar{\mathcal{F}},i)} : i \in [m]\}$ *where* $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ *and each* $\mathcal{F}_i$ *is the multilinear arithmetization of* F_i *obtained as defined in Proposition 4.3.*

We need some terminology. In the proof system, the oracle is expected to store a proof based on an interpretation: the proof should be a concatenation of strings $\{s_i : i \in [n]\}$ and $\{t_i : i \in [m]\}$ such that there exists an interpretation $\bar{F} = (F_1, \ldots, F_n)$ and its multilinear arithmetization $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ with $\{s_i\} = \{s^{(\bar{\mathcal{F}}_i)}\}$ and $\{t_i\} = \{t^{(\Psi,\bar{\mathcal{F}},i)}\}$. Note that for every s_i of length $N^m \cdot 2m \log N$, there always exists a polynomial $\mathcal{F}_i$ such that $s_i = s^{(\mathcal{F}_i)}$. Thus, for every oracle storing a concatenation of strings $\{s_i\}$ and $\{t_i\}$, we say that the oracle is based on polynomials $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ where each $\mathcal{F}_i$ is the polynomial with $s_i = s^{(\mathcal{F}_i)}$ and the verifier would interpret s_i and t_i as $s^{(\mathcal{F}_i)}$ and $t^{(\Psi,\bar{\mathcal{F}},i)}$ respectively. When the verifier queries for the $\bar{a}$-th entry of s_i for some $\bar{a} \in \mathcal{I}^m$, we say that it queries for $s_{\bar{a}}^{(\mathcal{F}_i)}$ or simply $\mathcal{F}_i(\bar{a})$. Likewise, when the verifier queries for the $\bar{a}$-th entry of t_i for some $\bar{a} \in \mathcal{I}^{i-1}$, we say that it queries for $t_{\bar{a}}^{(\Psi,\bar{\mathcal{F}},i)}$ or simply the coefficients of polynomial $p_i^{\Psi}(\bar{a}, x_i, \bar{\mathcal{F}})$.

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

Chapter 5 The PCP verifier for SAT(∃SOQBF)

In this chapter, we design a PCP verifier such that given an ∃SOQBF formula, the following holds.

- If the formula is satisfiable, then the verifier always accepts the oracle that stores the PCP proof constructed as in Chapter 4.
- Otherwise, the verifier rejects any oracle with high probability.

Let π be an oracle storing a concatenation of strings $\{s_i\}$ and $\{t_i\}$ which is based on polynomials $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$. To achieve the goal, the verifier does the following. First, it uses the multilinearity test [5, 19] to checks if each $\mathcal{F}_i$ is *close to* any multilinear function. Then, it uses the sum-check protocol [4, 5, 31, 42] to check if $\bar{\mathcal{F}}$ is an arithmetization of a satisfying interpretation for formula $\Psi(\bar{x}, \bar{f})$.

The organization is as follows. In Section 5.1, we introduce the multilinearity test which is adapted from [19]. In Section 5.2, we introduce the sum-check protocol in [5]. In Section 5.3, we construct the PCP verifier for SAT(∃SOQBF) and prove our main theorem in the thesis.

doi:10.6342/NTU202300853

5.1 The multilinearity test

In this section, we introduce the multilinearity test which could verify if each $\mathcal{F}_i$ is *close to* any multilinear function. Let ML be the set of multilinear functions from $\mathcal{I}^m$ to $\mathbb{Z}$. The notion of how close a function is to any multilinear function is defined as follows.

Definition 5.1. *For every function $F : \mathcal{I}^m \to \mathbb{Z}$, define $\Delta_{ML}(F)$ by*

$$\Delta_{ML}(F) \stackrel{def}{=} \max_{\mathcal{F} \in ML} \frac{|\{\bar{a} \in \mathcal{I}^m : F(\bar{a}) \neq \mathcal{F}(\bar{a})\}|}{N^m}$$

To verify if $\Delta_{\mathsf{ML}}(\mathcal{F}_i)$ is small or not, the multilinearity test checks if the values of $\mathcal{F}_i$ on some triples in a line are aligned. The definition of triples is as follows.

Definition 5.2. *A triple is a set of three distinct points $\bar{a}, \bar{b}, \bar{c} \in \mathbb{Z}^m$ where the points only differ in a single coordinate. For every function $\mathcal{F} : \mathbb{Z}^m \to \mathbb{Z}$, a triple $\{\bar{a}, \bar{b}, \bar{c}\}$ is $\mathcal{F}$-linear if there is a multilinear function $\mathcal{G} : \mathbb{Z}^m \to \mathbb{Z}$ such that $\mathcal{G}(\bar{a}) = \mathcal{F}(\bar{a}), \mathcal{G}(\bar{b}) = \mathcal{F}(\bar{b}), \mathcal{G}(\bar{c}) = \mathcal{F}(\bar{c})$.*

The idea of the test is that if a function $\mathcal{F}$ is far from any multilinear function, then a large fraction of triples are not aligned as well.

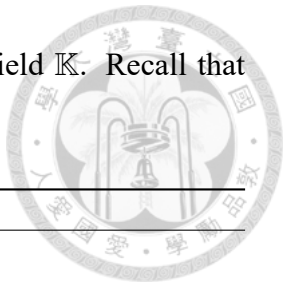
Theorem 5.3 ([19])**.** *For every function $\mathcal{F} : \mathcal{I}^m \to \mathbb{Z}$, if $\Delta_{ML}(\mathcal{F}) \geq \frac{1}{6\ell}$ for some ℓ, then at least $\frac{1}{50m\ell}$ fraction of triples in T are not $\mathcal{F}$-linear.*

Remark 5.4. *Although the exact bound stated above is not the one in [19], we could easily modify the proof in [19] to obtain the bound stated above.*

Let T be the set of triples whose points are in $\mathcal{I}^m$. Since each triple could be encoded

doi:10.6342/NTU202300853

as a binary string and $|T| < |\mathbb{K}| < 2|T|$, we could embed T into field $\mathbb{K}$. Recall that $H \subseteq \mathbb{K}$ with $|H| = 200m\ell$ The multilinearity test works as follows.



Algorithm 1 The multilinearity test

Ensure: ACCEPT if $\Delta_{\mathsf{ML}}(\mathcal{F}_i) < \frac{1}{6\ell}$ for each $i \in [n]$.

```
Let r_1, r_2 be random numbers in 𝕂.
Compute the set H' = {r_1 h + r_2 | h ∈ H}.
for h' ∈ H' do
    if h' could be decoded as a triple {ā, b̄, c̄} in 𝓘^m then
        Query the oracle for the values of F_i(ā), F_i(b̄), F_i(c̄) for every i ∈ [n].
        if the triple is not F_i-linear for some i or any value is larger than N^{2m} then
            REJECT
ACCEPT
```

Proposition 5.5. *For every $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ where each $\mathcal{F}_i : \mathcal{I}^m \to \mathbb{Z}$ is a function, performing the multilinearity test on an oracle with $\bar{\mathcal{F}}$ uses $O(m \log N)$ random bits, queries $O(nmN \log N)$ bits from the oracle, and runs in time $O(nmN \log^2 N \log\log N)$.*

Proof. The only randomness is used to generate r_1 and r_2 in $\mathbb{K}$, where the test uses $\log(|\mathbb{K}|) = O(m \log(mN))$ random bits. The test queries for the values of $\mathcal{F}_i(\bar{a}), \mathcal{F}_i(\bar{b}), \mathcal{F}_i(\bar{c})$ for each $i \in [n]$ each $h' \in H'$ that could be decoded as a triple $\{\bar{a}, \bar{b}, \bar{c}\}$. Since the answer of each query is a number less than N^{2m}, the test queries for at most $n|H| \log(N^{2m}) = O(nmN \log N)$ bits from the oracle. The test does multiplication and addition on number less than N^{2m}, which could be done in time $\log(N^{2m}) \log\log(N^{2m}) \log\log\log(N^{2m})) = O(m \log^2 N \log\log N)$ by Schönhage-Strassen algorithm [39]. Thus, the test runs in time $O(nmN \log^2 N \log\log N)$. □

Proposition 5.6. *For every $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ where each $\mathcal{F}_i : \mathcal{I}^m \to \mathbb{Z}$ is a function, the following holds.*

- *If each $\mathcal{F}_i$ is multilinear, then the test always accepts.*
- *If $\Delta_{\mathsf{ML}}(\mathcal{F}_i) \geq \frac{1}{6\ell}$ for some i, then the test rejects with probability at least $\frac{1}{2}$.*

doi:10.6342/NTU202300853

Proof. Clearly, by Remark 4.10, if each $\mathcal{F}_i$ is multilinear, then the test always accept.

Assume that $\Delta_{\mathsf{ML}}(\mathcal{F}_i) \geq \frac{1}{6\ell}$ for some i. Let $N \subset \mathbb{K}$ be the subset of non-$\mathcal{F}_i$-linear triples. Let $p = \frac{|N|}{|\mathbb{K}|}$. By $|\mathbb{K}| < 2|T|$ and Theorem 5.3,

$$p = \frac{|N|}{|\mathbb{K}|} > \frac{|N|}{2|T|} \geq \frac{1}{100m\ell} \tag{5.1}$$

For each $h \in H$, let I_h be the indicator variable that $r_1 h + r_2 \in N$. Let $I = \sum_{h \in H} I_h$. Then the probability that the test rejects is $\mathbf{Pr}_{r_1,r_2}[\, I > 0 \,]$. Note that for every $h, k \in \mathbb{K}$, $\mathbf{Pr}_{r_1,r_2}[\, r_1 h + r_2 = k \,] = \frac{1}{|\mathbb{K}|}$ and $\mathbf{Pr}_{r_1,r_2}[\, I_h = 1 \,] = \frac{|N|}{|\mathbb{K}|}$. By the linearity of expectation,

$$E[I] = \sum_{h \in H} E[I_h] = \sum_{h \in H} \mathbf{Pr}_{r_1,r_2}[\, I_h = 1 \,] = |H| \frac{|N|}{|\mathbb{K}|} = |H| p$$

Note that for every $h_1, h_2, k_1, k_2 \in \mathbb{K}$ with $h_1 \neq h_2$,

$$\mathbf{Pr}_{r_1,r_2}[\, r_1 h_1 + r_2 = k_1, r_1 h_2 + r_2 = k_2 \,] = \frac{1}{|\mathbb{K}|^2}$$

For every distinct $h_1, h_2 \in H$,

$$\begin{aligned} \mathbf{Pr}[\, I_{h_1} = I_{h_2} = 1 \,] &= \sum_{k_1,k_2 \in N} \mathbf{Pr}_{r_1,r_2}[\, r_1 h_1 + r_2 = k_1, r_1 h_2 + r_2 = k_2 \,] \\ &= |N|^2 \cdot \frac{1}{|\mathbb{K}|^2} \\ &= \mathbf{Pr}[\, I_{h_1} = 1 \,] \mathbf{Pr}[\, I_{h_2} = 1 \,] \end{aligned}$$

, which implies that I_{h_1} and I_{h_2} are independent. Thus,

$$\sigma^2(I) = \sum_{h \in H} \sigma^2(I_h) = |H| p (1 - p)$$

doi:10.6342/NTU202300853

By Chebyshev inequality and Equation 5.1,

$$\mathbf{Pr}[\ I \leq 0\ ] \leq \mathbf{Pr}[\ |I - E[I]| \geq E[I]\ ] \leq \frac{\sigma^2(I)}{(E[I])^2} = \frac{1-p}{|H|p} < \frac{1}{2}$$

□

5.2 The sum-check protocol

In this section, we introduce the sum-check protocol, which could check if $\bar{\mathcal{F}}$ is an arithmetization of a satisfying interpretation for formula $\Psi(\bar{x}, \bar{f})$. In particular, the protocol checks if Equation 4.1 holds for $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$. The protocol works as follows.

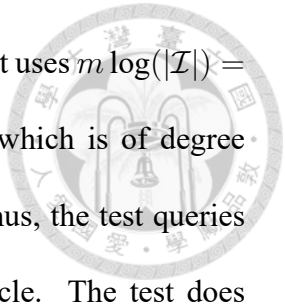
Algorithm 2 The sum-check protocol

Ensure: ACCEPT if Equation 4.1 holds.

Let $\bar{r} = (r_1, \ldots, r_m)$ be a random string in $\mathcal{I}^m$.
for $i = 1$ **to** m **do**
    Query the oracle for $p_i^{\Psi}(r_1, \ldots, r_{i-1}, x_i, \bar{\mathcal{F}})$ and denote the answer by $q_i(x_i)$.
    if $\deg(q_i)$ is larger than 2ℓ **or** any coefficient is larger than $N^{10m\ell}$ **then**
        REJECT
    else if $i = 1$ and $q_i(0) + q_i(1) \neq 0$ **then**
        REJECT
    else if $i > 1$ and $q_i(0) + q_i(1) \neq q_{i-1}(r_{i-1})$ **then**
        REJECT
Compute the formula $p_m^{\Psi}(\bar{x}, \bar{f})$ as defined in Proposition 4.4.
Compute the value of $p_m^{\Psi}(\bar{r}, \bar{\mathcal{F}})$ by querying the oracle for the values of $\mathcal{F}_i(\bar{a})$ that appears in $p_m^{\Psi}(\bar{r}, \bar{\mathcal{F}})$.
if $p_m^{\Psi}(\bar{r}, \bar{\mathcal{F}}) = q_m(r_m)$ **then**
    ACCEPT
else
    REJECT

Proposition 5.7. *For every $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ where each $\mathcal{F}_i : \mathcal{I}^m \to \mathbb{Z}$ is a function, the sum-check protocol on an oracle with $\bar{\mathcal{F}}$ and polynomials q_i uses $O(m \log N)$ random bits, queries $O(m^2 k^2 \log N)$ bits from the oracle, and runs in time $O(m^2 k^2 \log^2 N \log\log N)$.*

doi:10.6342/NTU202300853



Proof. The only randomness is used to generate $\bar{r}$ in $\mathcal{I}^m$, where the test uses $m \log(|\mathcal{I}|) = O(m \log N)$ random bits. The test queries for polynomials $q_i(x_i)$ which is of degree less than $O(k)$ and the coefficient of which is less than $O(N^{mk})$. Thus, the test queries for at most $O(mk \log(N^{mk})) = O(m^2k^2 \log N)$ bits from the oracle. The test does multiplication and addition on number less than $O(N^{mk})$, which could be done in time $\log(N^{mk}) \log\log(N^{mk}) \log\log\log(N^{mk})) = O(mk \log^2 N \log\log N)$ by Schönhage-Strassen algorithm [39]. Thus, the test runs in time $O(m^2k^2 \log^2 N \log\log N)$. □

The correctness of the protocol is based on the following proposition in algebra.

Proposition 5.8. *Every non-zero univariate polynomial of degree d has at most d roots.*

Proposition 5.9. *For every* ∃SOQBF *formula $\Psi(\bar{x}, \bar{f}) = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m \psi(\bar{x}, \bar{f})$, there is a verifier V such that*

- *If Equation 4.1 holds for some multilinear functions $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$, then for the oracle π that stores $\{s^{(\mathcal{F}_i)}\}$ and $\{t^{(\Psi,\bar{\mathcal{F}},i)}\}$, $\mathbf{Pr}_r[\, V^\pi(\Psi, r) = \mathsf{ACCEPT}\,] = 1$.*
- *If Equation 4.1 does not hold for any multilinear functions, then for any oracle that stores $\{s^{(\mathcal{F}_i)}\}$ and $\{t_i\}$ where $\Delta_{ML}(\mathcal{F}_i) \leq \frac{1}{6\ell}$ for each $i \in [n]$, $\mathbf{Pr}_r[\, V^\pi(\Psi, r) = \mathsf{ACCEPT}\,] < \frac{1}{2}$.*

Proof. Let V be the verifier that uses the sum-check protocol. In the i-th iteration of the for loop, we say that the oracle is honest if $q_i(x_i) = p_i^\Psi(r_1, \ldots, r_{i-1}, x_i, \bar{\mathcal{F}})$. Otherwise, we say that the oracle cheats. In the case that the oracle cheats in the i-th iteration, we say that the oracle is caught if $q_i(r_i) \neq p_i^\Psi(r_1, \ldots, r_i, \bar{\mathcal{F}})$ and the oracle escapes if otherwise.

Assume that Equation 4.1 holds for some multilinear functions $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$. Let π be the honest oracle that stores $\{s^{(\mathcal{F}_i)}\}$ and $\{t^{(\Psi,\bar{\mathcal{F}},i)}\}$. It is clear that verifier V with

doi:10.6342/NTU202300853

oracle access to π would always accept.

Assume that Equation 4.1 does not hold for any multilinear functions. Let π be an oracle that stores $\{s^{(\mathcal{F}_i)}\}$ and $\{t_i\}$ where $\Delta_{\mathsf{ML}}(\mathcal{F}_i) \leq \frac{1}{6\ell}$ for each $i \in [n]$. We first show that if each $\mathcal{F}_i$ is multilinear, then $\mathbf{Pr}_r[\, V^{\pi}(\Psi, r) = \mathsf{ACCEPT} \,] \leq \frac{2}{7}$. Then we show that if $\Delta_{\mathsf{ML}}(\mathcal{F}_i) \leq \frac{1}{6\ell}$ for each i, then $\mathbf{Pr}_r[\, V^{\pi}(\Psi, r) = \mathsf{ACCEPT} \,] < \frac{1}{2}$.

Case 1: each $\mathcal{F}_i$ is multilinear. Since Equation 4.1 does not hold for any multilinear functions, $p_1^{\Psi}(0) + p_1^{\Psi}(1) \neq 0$ and the oracle must cheat in the first iteration or else it would be rejected. In addition, if the oracle is caught in an iteration, i.e. $q_i(r_i) \neq p_i^{\Psi}(r_1, \ldots, r_i, \bar{\mathcal{F}})$, then it must cheat in the next iteration or else it would be rejected. The reason is that if it is honest in the next iteration, i.e. $q_{i+1}(x_{i+1}) = p_{i+1}^{\Psi}(r_1, \ldots, r_i, x_{i+1}, \bar{\mathcal{F}})$, then

$$\begin{aligned} q_{i+1}(0) + q_{i+1}(1) &= p_{i+1}^{\Psi}(r_1, \ldots, r_i, 0, \bar{\mathcal{F}}) + p_{i+1}^{\Psi}(r_1, \ldots, r_i, 1, \bar{\mathcal{F}}) \\ &= p_i^{\Psi}(r_1, \ldots, r_i, \bar{\mathcal{F}}) \\ &\neq q_i(r_i) \end{aligned}$$

By the definition of the protocol and Proposition 4.7, each $q_i(x_i)$ and each p_i^{Ψ} are of degree less than or equal to 2ℓ. Thus, by Proposition 5.8, the probability P_i that an oracle escapes in the i-th iteration conditional on it cheating in the previous iteration could be bounded as follows.

$$\begin{aligned} P_i &= \mathbf{Pr}_{r_i}[\, q_i(r_i) = p_i^{\Psi}(r_1, \ldots, r_i, \bar{\mathcal{F}}) \mid q_i(x_i) \neq p_i^{\Psi}(r_1, \ldots, r_{i-1}, x_i, \bar{\mathcal{F}}) \,] \\ &\leq \frac{\deg(q_i(x_i) - p_i^{\Psi}(r_1, \ldots, r_{i-1}, x_i, \bar{\mathcal{F}}))}{|\mathcal{I}|} \\ &\leq \frac{2\ell}{7m\ell} = \frac{2}{7m} \end{aligned}$$

doi:10.6342/NTU202300853

Therefore, the probability that the oracle escapes in some round conditional on cheating in the first iteration could be bounded by $m \cdot \frac{2}{7m} < \frac{2}{7}$. If the oracle is caught in the last iteration, i.e. $q_m(r_m) \neq p_{\Psi,m}(\bar{r}, \bar{\mathcal{F}})$, then it would be rejected after the verifier computes the value of $p_{\Psi,m}(\bar{r}, \bar{\mathcal{F}})$.

Case 2: $\Delta_{\mathsf{ML}}(\mathcal{F}_i) \leq \frac{1}{6\ell}$ for each i. Assume for contradiction that $\mathbf{Pr}_r[\, V^{\pi}(\Psi, r) = \mathsf{ACCEPT}\,] \geq \frac{1}{2}$. Let $\bar{\mathcal{G}} = (\mathcal{G}_1, \ldots, \mathcal{G}_n)$ be the multilinear functions such that $\Delta(\mathcal{F}_i, \mathcal{G}_i) < \frac{1}{6\ell}$ for each i. Let π' be the oracle that stores $\{s^{(\mathcal{G}_i)}\}$ and $\{t_i\}$. We show that $\mathbf{Pr}_r[\, V^{\pi'}(\Psi, r) = \mathsf{ACCEPT}\,] \geq \frac{2}{7}$, which contradicts with Case 1. Let I be the indicator that the answer of π to an query and that of π' differs. By the assumption that $\mathbf{Pr}_r[\, V^{\pi}(\Psi, r) = \mathsf{ACCEPT}\,] \geq \frac{1}{2}$, we have

$$\begin{aligned}
\mathbf{Pr}_r[\, V^{\pi'}(\Psi, r) = \mathsf{ACCEPT}\,] &\geq \mathbf{Pr}_r[\, V^{\pi'}(\Psi, r) = \mathsf{ACCEPT} \mid I = 0\,] \cdot \mathbf{Pr}_r[\, I = 0\,] \\
&= \mathbf{Pr}_r[\, V^{\pi}(\Psi, r) = \mathsf{ACCEPT}\,] \cdot \mathbf{Pr}_r[\, I = 0\,] \\
&\geq \frac{1}{2} \cdot \mathbf{Pr}_r[\, I = 0\,]
\end{aligned}$$

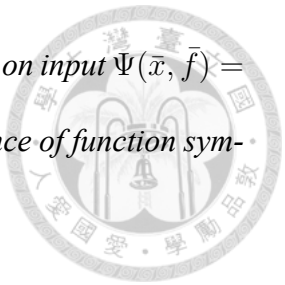
Since both oracles stores $\{t_i\}$, their answers to any query in each iteration are the same. They may answer differently to the queries of the values of $\mathcal{F}_i(\bar{y})$ and $\mathcal{G}_i(\bar{y})$ respectively. Since $\Delta(\mathcal{F}_i, \mathcal{G}_i) < \frac{1}{6\ell}$ and there are at most 2ℓ such queries, it follows that $\mathbf{Pr}_r[\, I = 1\,] \leq \frac{1}{3}$. Hence, $\mathbf{Pr}_r[\, V^{\pi'}(\Psi, r) = \mathsf{ACCEPT}\,] \geq \frac{1}{2} \cdot \frac{2}{3} \geq \frac{2}{7}$. □

5.3 The PCP verifier for SAT(∃SOQBF)

In this section, we construct the PCP verifier for SAT(∃SOQBF) by using the multilinearity test and the sum-check protocol.

Theorem 5.10. SAT(∃SOQBF) ∈ PCP$[n \log(n), n^4 \log(n)]$.

doi:10.6342/NTU202300853



In particular, there is a PCP verifier for SAT(∃SOQBF) *such that on input* $\Psi(\bar{x}, \bar{f}) = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m \psi(\bar{x}, \bar{f})$ *with* $|\psi| = k$ *and* ℓ *number of occurrence of function symbols in* ψ*, the following holds.*

(a) The verifier uses $O(m \log k)$ *random bits.*

(b) The verifier queries $O(m^2 k^2 \log k)$ *bits from the oracle.*

(c) The verifier runs in time $O(m^2 k^2 \log^3 k)$.

(d) The honest oracle answers each query in time $O(4^m m\ell^2 \log^2(\ell) \log\log(\ell))$ *(with a correct computation tableau).*

Proof. Let $\Psi(\bar{x}, \bar{f}) = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m \psi(\bar{x}, \bar{f})$ be an ∃SOQBF formula. We fix a set $\mathcal{I}$ of integers $\{0, 1, \ldots, N-1\}$ with $N = 7m\ell$. We also fix a finite field $\mathbb{K}$ with $mN^{m+2} < |\mathbb{K}| < 2mN^{m+2}$ and a subset $H \subset \mathbb{K}$ with $|H| = 200m\ell$. The PCP verifier for SAT(∃SOQBF) works as follows.

1. Compute $p^{\psi}(\bar{x}, \bar{f})$ and $p_m^{\Psi}(\bar{x}, \bar{f})$ as defined in Proposition 4.4 and Definition 4.6.
2. Perform the multilinearity test as defined in Algorithm 1.
3. Perform the sum-check protocol as defined in Algorithm 2.
4. Accept if and only if both the multilinearity test and the sum-check protocol accept.

By Proposition 5.5 and Proposition 5.7, the verifier uses $O(m \log N) = O(m \log k)$ random bits, queries $O(m^2 k^2 \log N)$ bits from the oracle, and runs in time $O(m^2 k^2 \log^3 k)$. By Proposition 4.11 and Proposition 4.14, the honest oracle could compute the answer to each query in $O(4^m m^2 k^2 \log^2 N \log\log N)$ time.

doi:10.6342/NTU202300853

Now we prove that the verifier works properly. That is, we prove the followings.

- If Ψ is satisfiable, then there is an oracle such that the verifier always accepts the oracle.
- If Ψ is not satisfiable, then for any oracle, the verifier rejects the oracle with probability at least $\frac{1}{2}$.

Assume that Ψ is satisfiable. By Proposition 4.8, there exist multilinear polynomials $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ such that Equation 4.1 holds. Let π be the oracle that stores $\{s^{(\mathcal{F}_i)} : i \in [n]\}$ and $\{t^{(\Psi,\bar{\mathcal{F}},i)} : i \in [m]\}$. By Proposition 5.6 and Proposition 5.9, the test and the protocol always accept oracle π. It follows that the verifier always accepts the oracle.

Assume that Ψ is not satisfiable. By Proposition 4.8, there do not exist multilinear polynomials $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$ such that Equation 4.1 holds. Let π be the oracle that stores $\{s_i\}$ and $\{t_i\}$ which is based on functions $\bar{\mathcal{F}} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$. Namely, for each $i \in [n]$, $s_i = s^{(\mathcal{F}_i)}$. If $\Delta_{\mathsf{ML}}(\mathcal{F}_i) \geq \frac{1}{6\ell}$ for some i, then by Proposition 5.6, the test would reject oracle π with probability at least $\frac{1}{2}$. If $\Delta_{\mathsf{ML}}(\mathcal{F}_i) < \frac{1}{6\ell}$ for each i, then by Proposition 5.9, the protocol would reject oracle π with probability at least $\frac{1}{2}$. In both cases, the verifier rejects the oracle with probability at least $\frac{1}{2}$. □

Corollary 5.11. *For every language L in $\textsc{Ntime}[2^{t(n)}]$ for some $t(n)$, there is a PCP verifier for L with the following properties.*

(a) The verifier uses $O(t(n)\log(t(n)))$ random bits.

(b) The verifier queries $O(t(n)^4\log(t(n)))$ bits from the oracle.

(c) The verifier runs in $O(t(n)^4\log^3(t(n)))$ time.

doi:10.6342/NTU202300853

(d) The honest oracle answers each query in $O(4^{t(n)}t^4(n)\log^2(t(n))\log\log(t(n)))$ *time (with a correct computation tableau).*

Proof. Let L be a language in $\text{NTIME}[2^{t(n)}]$ that is decided by non-deterministic Turing machine M. Let x be an input of length n. By Proposition 2.7, we could construct an $\exists$SOQBF formula $\Phi = \exists g_1 \dots \exists g_n \forall y_1 \dots \forall y_m \phi(\bar{y}, \bar{g})$ such that $x \in L$ if and only if Φ is satisfiable. By Remark 3.2, $n = O(1)$, $m = O(t(n))$, $\ell = O(1)$, and $k = O(t(n))$. By Theorem 5.10, there is a PCP verifier for L with the desired properties. □

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

Chapter 6 Conclusion

We list our main results and compare it with related works as follows. The first result is the inapproximability of SAT(∃SOQBF), which is obtained by the ∃SOQBF encoding of the computation.

Theorem 6.1 (The inapproximability of SAT(∃SOQBF))**.** *For every $\delta \in (0, 1)$, problem* δGAP-SAT(∃SOQBF) *is* NEXP*-hard.*

The second result is a PCP proof system for ∃SOQBF.

Theorem 6.2. SAT(∃SOQBF) $\in$ PCP$[n \log(n), n^4 \log(n)]$.

In particular, there is a PCP verifier for SAT(∃SOQBF) *such that on input $\Psi(\bar{x}, \bar{f}) = \exists f_1 \ldots \exists f_n \forall x_1 \ldots \forall x_m \psi(\bar{x}, \bar{f})$ with $|\psi| = k$ and ℓ number of occurrence of function symbols in ψ, the following holds.*

(a) The verifier uses $O(m \log k)$ random bits.

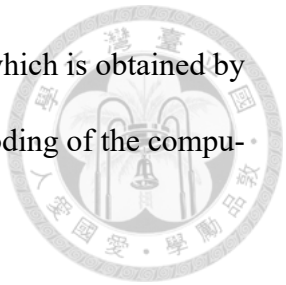
(b) The verifier queries $O(m^2 k^2 \log k)$ bits from the oracle.

(c) The verifier runs in time $O(m^2 k^2 \log^3 k)$.

(d) The honest oracle answers each query in time $O(4^m m \ell^2 \log^2(\ell) \log\log(\ell))$ (with a correct computation tableau).

doi:10.6342/NTU202300853

The third result is a PCP proof system for NEXP computation, which is obtained by combining the PCP proof system for ∃SOQBF and the ∃SOQBF encoding of the computation.



Corollary 6.3. *For every language L in $\text{NTIME}[2^{t(n)}]$ for some $t(n)$, there is a PCP verifier for L with the following properties.*

(a) The verifier uses $O(t(n)\log(t(n)))$ random bits.

(b) The verifier queries $O(t^4(n)\log(t(n)))$ bits from the oracle.

(c) The verifier runs in time $O(t^4(n)\log^3(t(n)))$.

(d) The honest oracle answers each query in time $O(4^{t(n)}t^4(n)\log^2(t(n))\log\log(t(n)))$ (with a correct computation tableau).

There are some advantages of our construction of the PCP proof system for NEXP computation, which are inherited from the reduction in [15]. First, our encoding of computation tableaux is natural in the sense that the encoding is essentially the Cook-Levin reduction [17, 30] disguised in the form of function certificates. Second, many concrete NEXP-complete problems [34] such as (succinct) 3-SAT, 3-colorability, Hamiltonian cycle, set packing and subset sum could be directly reduced to ∃SOQBF. In comparison with the reduction in [36], our reduction incurs less computation overhead. Thus, our PCP proofs for ∃SOQBF can be transformed as the PCP proofs for those practical problems. Third, it is easy to generalize our encoding for non-deterministic computation with any time complexity and space complexity, which might be useful to construct verifiable scheme for general non-deterministic computation with arbitrary time and space complexity. The following table is the comparison between our last result with related paper [5, 19] on a language $L \in \text{NTIME}[2^{t(n)}]$.

doi:10.6342/NTU202300853

	Babai et al. [5]	Feige et al. [19]	Our work
Random bits	$\mathsf{poly}(t(n))$	$t(n)\log t(n)$	$t(n)\log t(n)$
Query bits	$\mathsf{poly}(t(n))$	$t(n)\log t(n)$	$t^4(n)\log t(n)$
Time of verifier	$\mathsf{poly}(t(n))$	$\mathsf{poly}(2^{t(n)})$	$t^4(n)\log^3(t(n))$
Time of oracle	$\exp(t(n))$	$\exp(t(n))$	$\exp(t(n))$

Table 6.1: Comparison between different PCP proof systems for a language $L \in \text{NTIME}[2^{t(n)}]$

As for the future work, we hope to do the following. First, we would like to combine some cryptographic protocols with our protocol to construct a verifiable computation scheme for NEXP computation. Second, we want to use other logics such as dependency quantified boolean formula (DQBF) and Bernays-Schönfinkel-Ramsey (BSR) to encode the NEXP computation or other type of computation and find the best one with less overhead. Third, we are interested in showing other inapproximability results for other logics and succinctly represented problems in NEXP.

doi:10.6342/NTU202300853

doi:10.6342/NTU202300853

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

[2] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[3] L. Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429, 1985.

[4] L. Babai and L. Fortnow. Arithmetization: A new method in structural complexity theory. *computational complexity*, 1:41–66, 1991.

[5] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1:3–40, 1991.

[6] L. Babai and S. Moran. Arthur-merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[7] S. Badrinarayanan, Y. T. Kalai, D. Khurana, A. Sahai, and D. Wichs. Succinct del-

doi:10.6342/NTU202300853

egation for low-space non-deterministic computation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 709–721, 2018.

[8] V. Balabanov and J. Jiang. Reducing satisfiability and reachability to DQBF. *Austin, TX, USA, Sep*, 2015.

[9] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 113–131, New York, NY, USA, 1988. Association for Computing Machinery.

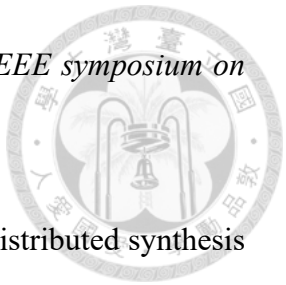
[10] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 90–108. Springer, 2013.

[11] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for r1cs. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 103–128. Springer, 2019.

[12] R. Bloem, R. Könighofer, and M. Seidl. SAT-based synthesis methods for safety specs. In *Verification, Model Checking, and Abstract Interpretation: 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings 15*, pages 1–20. Springer, 2014.

[13] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs:

doi:10.6342/NTU202300853



Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[14] K. Chatterjee, T. A. Henzinger, J. Otop, and A. Pavlogiannis. Distributed synthesis for LTL fragments. In *2013 Formal Methods in Computer-Aided Design*, pages 18–25. IEEE, 2013.

[15] F.-H. Chen, S.-C. Huang, Y.-C. Lu, and T. Tan. Reducing NEXP-complete problems to DQBF. In *CONFERENCE ON FORMAL METHODS IN COMPUTER-AIDED DESIGN–FMCAD 2022*, page 199, 2022.

[16] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 738–768. Springer, 2020.

[17] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[18] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112, 2012.

[19] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*, 43(2):268–292, 1996.

[20] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.

doi:10.6342/NTU202300853

[21] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 465–482. Springer, 2010.

[22] I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, volume 16, 2016.

[23] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker. Equivalence checking of partial designs using dependency quantified boolean formulae. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 396–403. IEEE, 2013.

[24] O. Goldreich. Short locally testable codes and proofs. *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 333–372, 2011.

[25] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost-linear length. *Journal of the ACM (JACM)*, 53(4):558–655, 2006.

[26] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304. Association for Computing Machinery, New York, NY, USA, 1985.

[27] J. Holmgren and R. Rothblum. Delegating computations with (almost) minimal time

doi:10.6342/NTU202300853

and space overhead. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–135. IEEE, 2018.

[28] J.-H. R. Jiang. Quantifier elimination via functional composition. In *Computer Aided Verification: 21st International Conference, CAV 2009, Grenoble, France, June 26-July 2, 2009. Proceedings 21*, pages 383–397. Springer, 2009.

[29] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1377–1394, 2002.

[30] L. A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.

[31] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

[32] S. Micali. Cs proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 436–453. IEEE, 1994.

[33] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 229–234, 1988.

[34] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Inf. Control.*, 71:181–185, 1986.

[35] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

doi:10.6342/NTU202300853

[36] G. L. Peterson and J. H. Reif. Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 348–363. IEEE, 1979.

[37] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.

[38] C. Scholl and B. Becker. Checking equivalence for partial implementations. In *Proceedings of the 38th Annual Design Automation Conference*, pages 238–243, 2001.

[39] A. Schönhage. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.

[40] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84, 2013.

[41] S. T. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, volume 1, page 17, 2012.

[42] A. Shamir. IP=PSPACE. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.

[43] Y. Tauman Kalai, R. Raz, and R. D. Rothblum. Delegation for bounded space. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 565–574, 2013.

[44] L. Trevisan. Inapproximability of combinatorial optimization problems. *Paradigms of Combinatorial Optimization: Problems and New Approaches*, pages 381–434, 2014.

doi:10.6342/NTU202300853

[45] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, 2015.

doi:10.6342/NTU202300853