Succinct Non-Subsequence Arguments

San Ling[1], Khai Hanh Tang[1(✉)], Khu Vu[2], Huaxiong Wang[1], and Yingfei Yan[3]

[1] Nanyang Technological University, 50 Nanyang Ave, Singapore
{lingsan,khaihanh.tang,hxwang}@ntu.edu.sg
[2] National University of Singapore, 21 Lower Kent Ridge Road, Singapore
isevvk@nus.edu.sg
[3] Xidian University, Xi'an, China
yanxi@stu.edu.xidian.cn

Abstract. Lookup arguments have recently attracted a lot of developments due to their applications in the constructions of succinct non-interactive arguments of knowledge (SNARKs). A closely related topic is subsequence arguments in which one can prove that string $\mathbf{s}$ is a subsequence of another string $\mathbf{t}$, i.e., deleting some characters in $\mathbf{t}$ can achieve $\mathbf{s}$. A dual notion, namely, non-subsequence arguments, is to prove that $\mathbf{s}$ is not a subsequence of $\mathbf{t}$. These problems have a lot of important applications in DNA sequence analysis, internet of things, blockchains, natural language processing, speech recognition, etc. However, despite their applications, they are not well-studied in cryptography, especially succinct arguments for non-subsequences with efficient proving time and sublinear verification time.

In this work, we propose the first succinct non-subsequence argument. Our solution applies the sumcheck protocol and is instantiable by any multivariate polynomial commitment schemes (PCSs). We achieve an efficient prover whose running time is linear in the size of sequences $\mathbf{s}$, $\mathbf{t}$ and their respective alphabet Σ. Our proof is succinct and the verifier time is sublinear assuming the employed PCS has succinct commitments and sublinear verification time. When instantiating with Sona PCS (EUROCRYPT'24), we achieve proof size $\mathcal{O}(\log_2|\mathbf{s}| + \log_2|\mathbf{t}| + \log_2|\Sigma|)$, prover time $\mathcal{O}(|\mathbf{s}| + |\mathbf{t}| + |\Sigma|)$ and verifier time $\mathcal{O}(\sqrt{|\mathbf{s}|} + \sqrt{|\mathbf{t}|} + \sqrt{|\Sigma|})$.

Extending our technique, we can achieve a batch subsequence argument for proving in batch k interleaving subsequence and non-subsequence arguments without proof size suffering a linear blow-up in k.

Keywords: SNARK · sumcheck · lookup

1 Introduction

SNARKs. Zero-knowledge proofs [13] allow a party to convince another party about the truth of some statement without leaking anything beyond its validity. During their history, zero-knowledge proofs have been developed in multiple aspects for multiple purposes. Among them, succinct non-interactive arguments

of knowledge (SNARKs) [12,11] are a kind of zero-knowledge proof that is non-interactive and has sublinear proof size. SNARKs have been employed to prove various problems including general NP statements [21] and incrementally verifiable computation (IVC) [16,19]. One way to achieve SNARKs is to construct polynomial IOP (PIOP) [4] from sumcheck arguments [17]. Then, we compile PIOP by any multilinear polynomial commitment schemes (PCSs) [15,25,22] with succinct commitment and (possibly) sublinear verifier time.

Lookup, Subsequence, and Non-Subsequence Arguments. One of the famous problems in SNARKs is proving lookup arguments. Initially introduced by Bootle et al. [3], lookup arguments aim to prove that a length-n vector $\mathbf{s} = (s_1, \ldots, s_n)$ is contained in a lookup table $\mathbf{t} = (t_1, \ldots, t_N)$, i.e., each s_j belongs to $\{t_i\}_{i\in[N]}$. Recent lookup arguments, cq [7], $\mathsf{cq}^+, \mathsf{cq}^{++}$ [5] and Locq [28] achieved fast lookup arguments by pushing large amount of computing into pre-precessing. The core of these results is Haböck's logarithmic derivatives [14]. Haböck's technique is suitable with sumcheck, avoids rearranging elements of $\mathbf{s}$ and $\mathbf{t}$ as in Plookup [9], achieves constant proof size, linear prover time (in the total length of $\mathbf{s}$ and $\mathbf{t}$) and sublinear verifier time. Another lookup argument, namely, Lasso [22], aims to optimize efficiency when the lookup table is structured.

A closely related problem to lookup is the subsequence argument. Here, a string $\mathbf{s}$ is a subsequence of $\mathbf{t}$ if elements in $\mathbf{s}$ appear in $\mathbf{t}$ in the same relative order. Equivalently saying, $\mathbf{s}$ is a subsequence of $\mathbf{t}$ if we can delete some characters in $\mathbf{t}$ to achieve $\mathbf{s}$. Besides, if $\mathbf{s}$ is not a subsequence of $\mathbf{t}$, we call this case non-subsequences. Arguments for subsequences and non-subsequences have a large number of applications in various fields including DNA sequence analysis, internet of things (IoT), blockchains, natural language processing, speech recognition, computer vision, and financial analysis [10]. We list below a few specific examples.

- In DNA sequence analysis, (non-)subsequence arguments show whether a sequence $\mathbf{s}$, of DNA characters A, T, G, and C, is a subsequence of another sequence $\mathbf{t}$, revealing some relationship between two species. In IoT, we can deploy health care systems for determining whether a DNA sequence of a particular person is vulnerable to some kind of disease. In some situations, this can be done by showing the person's DNA is a subsequence/non-subsequence of some other DNA sequences related to some disease.
- In blockchains, stored transactions are usually huge. Nowadays, some applications/smart contracts in blockchains may involve supporting illegal activities. For example, Tornado Cash can facilitate money-laundering activities. A person can prove himself not involved in such illegal activities by showing that his transaction history in a specific blockchain is a non-subsequence of transactions related to those potentially supporting-illegal-activities applications/smart contracts.
- In natural language processing, one can determine whether a list of words is a subsequence in a large (encoded) text, identifying the necessary properties, e.g., languages, topics (history, geography, or mathematics), etc.

– In speech recognition, (non-)subsequence arguments show whether a sequence of phonemes is a subsequence in sounds.

Although there are various applications, when putting them into the context of cryptography, they are not well-studied, especially SNARKs for subsequences and non-subsequences with efficient running time. Since data about DNA, languages, and sound is usually large, using SNARKs for proving and verifying the above examples would achieve great benefits, especially for verifiers with little resources of computations. To our knowledge, the only result from Thakur [23] proposes a SNARK for subsequences with sublinear verification time. However, there is no result proposing non-subsequence arguments with efficient running time and succinct proof size. Moreover, constructing SNARKs for non-subsequences is not a trivial task. To check whether length-n string $\mathbf{s}$ is not a subsequence of a length-N string $\mathbf{t}$, a trivial way is to check that all possible length-n subsequences of $\mathbf{t}$ do not match $\mathbf{s}$. This checking costs an exponential blow-up in running time as there are $\binom{N}{n}$ such subsequences in $\mathbf{t}$. Thus, we are asking the following question.

Can we construct succinct non-subsequence arguments with efficient proving time and sublinear verification time?

1.1 Our Contributions

In this result, we answer the above question by proposing the first construction of succinct non-subsequence arguments for proving that $\mathbf{s}$ is not a subsequence of $\mathbf{t}$. Our construction is a PIOP [4] from sumcheck and combine it with any multivariate polynomial commitment scheme (PCS) to achieve SNARKs. We achieve a proof size bounded by the total commitment sizes and evaluation proof sizes of related polynomials. The prover's running time is linear in N, n, $|\Sigma|$, and the total committing and evaluating time, while the verifier's running time is bounded by the verification time of evaluations of polynomials. If the employed PCS is secure and has succinct proof size and sublinear verifier time, then our construction achieves a SNARK for non-subsequences. Specifically, when using lookup arguments adapted from Haböck [14] and Sona PCS from [22] we achieve succinct non-subsequence arguments with proof size $\mathcal{O}(\log_2 n + \log_2 N + \log_2 |\Sigma|)$, linear prover time $\mathcal{O}(n+N+|\Sigma|)$ and sublinear verifier time $\mathcal{O}(\sqrt{n}+\sqrt{N}+\sqrt{|\Sigma|})$ (see Section 4.4).

Moreover, of independent interest, we briefly discuss how to prove batch subsequence arguments by showing in batch k interleaving subsequence and non-subsequence arguments without proof size suffering a linear blow-up in k.

1.2 Technical Overview

Before discussing the technical overview, we formally define subsequences and non-subsequences as follows. For $\mathbf{s} = (s_j)_{j\in[n]}$ and $\mathbf{t} = (t_j)_{j\in[N]}$, $\mathbf{s}$ is a subsequence of $\mathbf{t}$, denoted by $\mathbf{s} \lhd \mathbf{t}$, if there exists $\mathsf{id}_1, \ldots, \mathsf{id}_n$ s.t. $1 \leq \mathsf{id}_1 < \cdots <$

SubseqSearch :

Inputs: $n, N \in \mathbb{Z}_+$, alphabet Σ and $\mathbf{s} \in \Sigma^n$, $\mathbf{t} \in \Sigma^N$.
Goal: Determine whether $\mathbf{s} \lhd \mathbf{t}$, i.e., whether $\mathbf{s}$ is a subsequence of $\mathbf{t}$.
Execution: This algorithm works as follows:

1. Set $p_0 := 0$ and $s_{n+1} := \perp$.
2. For j from $1, \ldots, N$: If $s_{p_{j-1}+1} = t_j$, set $p_j := p_{j-1} + 1$; else, set $p_j := p_{j-1}$.
3. Return 1 indicating $\mathbf{s} \lhd \mathbf{t}$, if $p_N = n$; and 0, otherwise.

Fig. 1. Algorithm SubseqSearch.

$\mathsf{id}_n \leq N$ and $t_{\mathsf{id}_j} = s_j \; \forall j \in [n]$. For non-subsequence, $\mathbf{s}$ is not a subsequence of $\mathbf{t}$, denoted by $\mathbf{s} \not\lhd \mathbf{t}$, if no such a sequence $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$ exits. For non-subsequence, if $\mathbf{s}$ is not a subsequence of $\mathbf{t}$, we write $\mathbf{s} \not\lhd \mathbf{t}$.

We now discuss the technical overview. First, we recall the subsequence argument from Thakur [23] and discuss its exponential blow-up in the prover's running time. Then, we provide our technique, which can be plugged by any PCS and lookup arguments, to avoid such mentioned blow-up. At last, we discuss how to achieve batch subsequence arguments.

Subsequence Argument from Thakur [23]. The construction proves that $\mathbf{s} \lhd \mathbf{t}$ by showing that the indices corresponding to each element of $\mathbf{s}$ in $\mathbf{t}$ are in increasing order. Let $\mathbf{s} = (s_j)_{j\in[n]}$ and $\mathbf{t} = (t_j)_{j\in[N]}$ be strings of length n and N, respectively. The prover convinces that $\mathbf{s} \lhd \mathbf{t}$ by proving the existence of $(\mathsf{id}_j)_{j\in[n]}$ s.t. $\{(\mathsf{id}_j, s_j)\}_{j\in[n]} \subseteq \{(j, t_j)\}_{j\in[N]}$ and $(\mathsf{id}_j)_{j\in[n]}$ is increasing, namely, $\mathsf{id}_1 < \cdots < \mathsf{id}_n$. To show the latter, one can prove that $(\mathsf{id}_j - \mathsf{id}_{j-1})_{j\in[2,n]}$ is a sequence of positive integers.

Our Non-Subsequence Argument. To prove $\mathbf{s} \not\lhd \mathbf{t}$, using the above technique would require considering all $\binom{N}{n}$ possibilities for $(\mathsf{id}_j)_{j\in[n]}$. However, $\binom{N}{n}$ is exponential in N. To remove this exponential blow-up, we use algorithm SubseqSearch, described in Figure 1, to determine whether $\mathbf{s}$ is a subsequence of $\mathbf{t}$. Then, we can achieve the non-subsequence arguments by proving its correct execution.

The algorithm SubseqSearch tracks the maximal subsequence positions of $\mathbf{s}$ within $\mathbf{t}$ by looking for a sequence $(p_j)_{j\in[0,N]}$ s.t. $p_j \in [n]$ is the maximum index satisfying $(s_1, \ldots, s_{p_j}) \lhd (t_1, \ldots, t_j)$. If p_N computed from SubseqSearch equals to n, namely, the length of $\mathbf{s}$, it means $(s_1, \ldots, s_{p_N}) = (s_1, \ldots, s_n) \lhd \mathbf{t}$. Otherwise, when $p_N < n$, it implies $\mathbf{s} \not\lhd \mathbf{t}$. The correctness of this algorithm is based on the following Lemma 1 whose proof is deferred to Appendix A.

Lemma 1 (Correctness of Algorithm SubseqSearch). *Let $n, N \in \mathbb{Z}_+$, Σ be an alphabet, $\mathbf{s} \in \Sigma^n$ and $\mathbf{t} \in \Sigma^N$. Let $s_{n+1} := \perp$ where $\perp \notin \Sigma$. Let $(p_j)_{j\in[0,N]}$ satisfy that $p_0 = 0$ and, for $j \in [N]$, p_j is the maximum index satisfying $(s_1, \ldots, s_{p_j}) \lhd (t_1, \ldots, t_j)$. Then, for $j \in [N]$, $p_j = p_{j-1} + 1$, if $s_{p_{j-1}+1} = t_j$, and $p_j = p_{j-1}$, otherwise.*

Having $(p_j)_{j\in[0,N]}$ computed from algorithm SubseqSearch, for non-subsequence arguments, the prover needs to prove that $(p_j)_{j\in[0,N]}$ is computed correctly and $p_N < n$ as specified in Figure 1. To this end, we reduce all related constraints, in-

volving computing $(p_j)_{j\in[N]}$ from $\mathbf{s}$ and $\mathbf{t}$ into an equivalence system containing only lookup arguments as follows.

From Lemma 1, for $j \in [N]$, the relationship between p_j and p_{j-1} is either

$$p_j - p_{j-1} = 1 \iff s_{p_{j-1}+1} = t_j \text{ or } p_j - p_{j-1} = 0 \iff s_{p_{j-1}+1} \neq t_j. \quad (1)$$

Therefore, to capture this equivalence, we define the set $\mathsf{enc}(\mathbf{s}_\perp)$ containing elements of the form $(c, \ell, v) \in \{0,1\} \times [n+1] \times \Sigma \cup \{\perp\}$ s.t. $c = 1$ iff $s_\ell = v$. Specifically, c is the predicate capturing whether the ℓ-th entry of $\mathbf{s}$ is equal to v. Then, we can capture equivalence (1) by proving that $(p_j - p_{j-1}, p_{j-1}+1, t_j) \in \mathsf{enc}(\mathbf{s}_\perp)$ for $j \in [N]$. Hence, (1) holds iff $(p_j - p_{j-1}, p_{j-1} + 1, t_j) \in \mathsf{enc}(\mathbf{s}_\perp)$ for $j \in [N]$.

However, $\mathsf{enc}(\mathbf{s}_\perp)$ has $(n+1)\cdot(|\Sigma|+1)$ elements. This requires prover to work in time $\mathcal{O}(n \cdot |\Sigma|)$ for constructing it. Moreover, in the proof, the prover needs to capture the correct formation of $\mathsf{enc}(\mathbf{s}_\perp)$ as we expect verifier time to be sublinear without verifying it in the clear. To avoid this inefficiency, we devise the following solution. For $j \in [N]$, as computing p_j is based on p_{j-1} and the condition whether $s_{p_{j-1}+1} = t_j$, we introduce $v_j := s_{p_{j-1}+1}$ with an associated proof that v_j is correctly computed. This proof can be proceeded by proving

$$\{(p_{j-1}+1, v_j)\}_{j\in[N]} \subseteq \{(j, s_j)\}_{j\in[n+1]}. \quad (2)$$

With v_j, prover can determine whether $s_{p_{j-1}+1} = t_j$ or not by simply comparing t_j and v_j. Let $c_j = (t_j = v_j)$, namely, $c_j = 1$, if $t_j = v_j$ (or, $t_j - v_j = 0$) and $c_j = 0$, otherwise. To show that c_j is correctly computed, we assume that $\Sigma \cup \{\perp\} = \{i\}_{i\in[0,|\Sigma|]}$, namely, a set of $|\Sigma|+1$ elements from 0 to $|\Sigma|$. One sees that $t_j - v_j \in [-|\Sigma|, |\Sigma|]$. Therefore, the prover shows that

$$\{(c_j, t_j - v_j)\}_{j\in[N]} \subseteq \{(0,i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1,0)\} \quad (3)$$

which captures the well-formedness of $(c_j)_{j\in[N]}$. In fact, $c_j = 0$ iff $t_j - v_j \in [-|\Sigma|, |\Sigma|] \setminus \{0\}$. Finally, with c_j indicating whether $s_{p_{j-1}+1} = t_j$, we can then prove the relationship between p_j and p_{j-1} via proving that

$$p_j - p_{j-1} = c_j \quad \forall j \in [N] \quad (4)$$

according to (1). Hence, we removed the inefficiency discussed above by showing the two tuple lookup arguments (2) and (3). When using appropriate lookup argument (e.g., [14]) from sumcheck, prover time is only $\mathcal{O}(N + n + |\Sigma|)$ which does not contain factor $\mathcal{O}(n \cdot |\Sigma|)$ incurred by $\mathsf{enc}(\mathbf{s}_\perp)$ discussed above.

Finally, we achieve the following Theorem 1, an informal version of Theorem 2, for capturing subsequences and non-subsequences. In this theorem, we directly replace c_j by $p_j - p_{j-1}$ for $j \in [N]$ without the need to prove $c_j = p_j - p_{j-1}$.

Theorem 1 (Informal Version of Theorem 2). $\mathbf{s} \not\lhd \mathbf{t}$ *(respectively,* $\mathbf{s} \lhd \mathbf{t}$*) iff there exist* s_{n+1}, $(p_j)_{j\in[0,N]}$, *with* $p_0 = 0$ *by default, and* $\mathbf{v} = (v_j)_{j\in[N]}$ *satisfying*

$$\begin{cases} \{s_j\}_{j\in[n]} \subseteq \Sigma, \quad s_{n+1} = \perp, \quad \{t_j\}_{j\in[N]} \subseteq \Sigma, \\ \{(p_{j-1}+1, v_j)\}_{j\in[N]} \subseteq \{(j, s_j)\}_{j\in[n+1]}, \\ \{(p_j - p_{j-1}, t_j - v_j)\}_{j\in[N]} \subseteq \{(0,i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1,0)\}, \end{cases}$$

and $p_N < n$ *(respectively,* $p_N = n$*).*

Hence, we can construct succinct subsequence arguments from sumcheck based on (lookup) constraints in Theorem 1. Since there are various choices for lookup arguments and PCSs, for simplicity, we choose lookup arguments adapted from Haböck [14] as running time is linear in $\mathcal{O}(n+N+|\Sigma|)$ or any other lookup argument with similar running time. As Theorem 1 requires computing $(p_j)_{j\in[N]}$ when $\mathbf{s}$ is known, we cannot apply pre-processed lookup arguments like cq [7] because running time will be dominated by $\mathcal{O}(N \cdot \log_2 N)$. For PCS, we can instantiate by Sona [22] with constant proof size, linear prover time, and sublinear verifier time (see Section 4.4 for efficiency of the final protocol).

Extension to Batch Subsequence Arguments. We briefly discuss how the approach above can be extended to batch subsequence arguments with the following context. Given $k \in \mathbb{Z}_+$, $\{\mathbf{s}^{(i)}\}_{i\in[k]}$, $\{\mathbf{t}^{(i)}\}_{i\in[k]}$ and $\{b^{(i)}\}_{i\in[k]}$, prover shows that $b^{(i)} = \mathbf{s}^{(i)} \lhd \mathbf{t}^{(i)}$ for $i \in [k]$, i.e., $b^{(i)} = 1$, if $\mathbf{s}^{(i)} \lhd \mathbf{t}^{(i)}$, and $b^{(i)} = 0$, otherwise. Although SNARKs, especially sumcheck arguments, are well-known for batch arguments, i.e., by aggregating and proving all k arguments at once, proving such batch subsequence arguments is non-trivial. We observe that, if the verifier knows exactly each bit in $\{b^{(i)}\}_{i\in[k]}$, i.e., the verifier can distinguish subsequence and non-subsequence arguments for each $i \in [k]$, then, to optimize prover time, prover and verifier proceed either Thakur's approach for $b^{(i)} = 1$ (subsequence) or approach from Theorem 1 for $b^{(i)} = 0$ (non-subsequence). Arguably, Thakur's approach is better in efficiency for prover time when proving subsequence arguments. However, this way requires the verifier to know entirely $\{b^{(i)}\}_{i\in[k]}$ which requires a linear blow-up in the communication if k is large.

Assume that k is a power of 2. To reduce communication, we can encode $\{b^{(i)}\}_{i\in[k]}$ into a polynomial $\tilde{f}_b(\boldsymbol{X}) \in \mathbb{F}[l_k]$ where $l_k = \log_2 k$. Then, prover can commit to $\tilde{f}_b(\boldsymbol{X})$ to obtain and send commitment $\sigma(\tilde{f}_b)$ to verifier. If $\sigma(\tilde{f}_b)$ is a succinct commitment, verifier cannot know entire $\{b^{(i)}\}_{i\in[k]}$ even if $\sigma(\tilde{f}_b)$ is hiding or not. Therefore, the prover and verifier cannot proceed batch arguments with both methods from Thakur and Theorem 1 as they are not compatible. To avoid such incompatibility, let prover and verifier follow Theorem 1 to prove the well-formedness of sequence $(p_1^{(i)}, \ldots, p_N^{(i)})$, for $i \in [k]$, from running algorithm $\mathsf{SubseqSearch}$ on inputs $\mathbf{s}^{(i)}$ and $\mathbf{t}^{(i)}$ for the i-th (non-)subsequence argument. Hence, for $b^{(i)}$ equal to either 0 or 1, we prove the same way for well-formedness of such sequence $(p_1^{(i)}, \ldots, p_N^{(i)})$. Consequently, we can make a batch proof for well-formedness of all $(p_1^{(i)}, \ldots, p_N^{(i)})$ for $i \in [k]$. The only difference is showing that, for $i \in [k]$, either $p_N^{(i)} < n$ if $b^{(i)} = 0$ or $p_N^{(i)} = n$ if $b^{(i)} = 1$. These can be proven in batch by proving the tuple lookup argument $\{(b^{(i)}, p_N^{(i)})\}_{i\in[k]} \subseteq \{(0, j)\}_{j\in[0,n-1]} \cup \{(1, n)\}$. Specifically, by writing $\mathbf{s}^{(i)} = (s_1^{(i)}, \ldots, s_n^{(i)})$, $\mathbf{t}^{(i)} = (t_1^{(i)}, \ldots, t_N^{(i)})$ and extending Theorem 1, we can construct this batch argument by proving knowledge of $(s_{n+1}^{(i)})_{i\in[k]}$, $(p_j^{(i)})_{i\in[k],j\in[N]}$ and $(v_j^{(i)})_{i\in[k],j\in[N]}$ satisfying

$$\begin{cases} \{s_j^{(i)}\}_{i\in[k],j\in[n]} \subseteq \Sigma, \quad s_{n+1}^{(i)} = \perp \ \ \forall i \in [k], \quad \{t_j^{(i)}\}_{i\in[k],j\in[N]} \subseteq \Sigma, \\ \{(i, p_{j-1}^{(i)} + 1, v_j^{(i)})\}_{i\in[k],j\in[N]} \subseteq \{(i, j, s_j^{(i)})\}_{i\in[k],j\in[n+1]}, \\ \{(p_j^{(i)} - p_{j-1}^{(i)}, t_j^{(i)} - v_j^{(i)})\}_{i\in[k],j\in[N]} \subseteq \{(0, i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1, 0)\}, \\ \{(b^{(i)}, p_N^{(i)})\}_{i\in[k]} \subseteq \{(0, j)\}_{j\in[0,n-1]} \cup \{(1, n)\}. \end{cases}$$

This system contains only four lookup arguments. The first three lines of the above system are the extension of Theorem 1 with associated index i indicating the i-th (non-)subsequence argument. The last line is already explained above.

1.3 Related Works

Besides lookup arguments and subsequence arguments from Thakur [23], another line of research [18,20,26,2] is to prove correct execution of a regex searching. A regular expression (regex) searching algorithm uses a regex, such as keywords, to look for the appearance of such patterns in a long string, like documents. The general methodology requires transforming the regex into a graph-like structure, such as non-deterministic finite automata (NFA) or related notions like SAFA [2], TNFA [24] and ADFA [1]. To construct a non-subsequence argument, i.e., to prove $\mathbf{s} \not\triangleleft \mathbf{t}$, one can transform the sequence $\mathbf{s}$ into a regex. (For example, if $\mathbf{s} =$ "abac", we can transform into "$.*\mathsf{a}.*\mathsf{b}.*\mathsf{a}.*\mathsf{c}.*$".) Then, prove that the regex does not match the sequence $\mathbf{t}$. However, this approach leads to a complicated proof. In terms of efficiency, [18,26] achieve $\mathcal{O}(N \cdot |\mathbf{Q}_{\mathsf{TNFA}}|)$ constraints where $\mathbf{Q}_{\mathsf{TNFA}}$ is the description of the TNFA [24], [20] achieves $\mathcal{O}(N + |\mathbf{Q}_{\mathsf{ADFA}}|)$ where $\mathbf{Q}_{\mathsf{ADFA}}$ is the description of ADFA [1] with size $\mathcal{O}(n \cdot |\Sigma|)$, Reef [2] with SAFA achieves the resulting number of constraints at least $\mathcal{O}(N \cdot \log_2 N)$.

From Section 1.2, our approach reduces to lookups with sets of sizes bounded by $\mathcal{O}(n + N + |\Sigma|)$, achieving prover time $\mathcal{O}(n + N + |\Sigma|)$. Hence, we adopt a different approach whose running time outperforms those for proving regexes.

2 Preliminaries

2.1 Notations

Denote by $\mathbb{F}$, $\mathbb{Z}$ and $\mathbb{Z}_+$ to be the finite field of prime order, the ring of integers, and the set of non-negative integers, respectively. For any $a, b \in \mathbb{Z}$ satisfying $a \leq b$, denote by $[a, b]$ the set $\{a, a+1, \ldots, b\}$. When $a = 1$, we write $[b]$ to indicate $[1, b]$. For a finite set $\mathcal{S}$, denote by $|\mathcal{S}|$ to indicate its cardinality. Given $n \in \mathbb{Z}_+$, let $(s_i)_{i\in[n]} = (s_1, \ldots, s_n)$ denote a vector of length n and $\{s_i\}_{i\in[n]} = \{s_1, \ldots, s_n\}$ denote a set of size n. We write $\mathsf{bin}_l(a) = (b_1, \ldots, b_l) \in \{0, 1\}^l$ to indicate the binary representation of $a \in \mathbb{Z}_+$ in l bits s.t. $a = \sum_{i\in[l]} 2^{i-1} \cdot b_i$. Given $l \in \mathbb{Z}_+$ and $\mathbf{b} = (b_1, \ldots, b_l) \in \{0, 1\}^l$, define $\mathsf{int}_l(\mathbf{b}) = \sum_{i\in[l]} 2^{i-1} \cdot b_i$. We usually write $\mathsf{negl}(\lambda)$ to indicate the existence of a function negligible in λ.

Regarding multivariate polynomials, for $l \in \mathbb{Z}_+$, we denote by $\mathbb{F}[l]$ the set of l-variate polynomials over $\mathbb{F}$. A polynomial $f(\boldsymbol{X}) \in \mathbb{F}[l]$ is of degree ρ if the maximum of individual variables' degrees in $f(\boldsymbol{X})$ is ρ.

2.2 Succinct Non-Interactive Arguments of Knowledge (SNARKs)

A succinct non-interactive argument of knowledge (SNARK) for an NP relation $(x, w) \in \mathcal{R}$ is a non-interactive proof produced by a prover $\mathcal{P}$ and verified by a verifier $\mathcal{V}$. A SNARK should satisfy completeness, knowledge soundness, and succinctness as below. Detailed properties are in Appendix B.1. In this result, we are interested in SNARKs with sublinear verification time.

- *Completeness.* An honest prover $\mathcal{P}$ will always be accepted by $\mathcal{V}$.
- *Knowledge Soundness.* If a malicious prover $\mathcal{P}^*$ outputs an accepted proof π for the statement x, one can extract the witness w^* s.t. $(x, w^*) \in \mathcal{R}$.
- *Succinctness.* The proof size is sublinear in the size of the statement x.
- *Sublinear Verification Time.* The verification time is sublinear in $|x|$.

2.3 Multilinear Extension (MLE)

We recall the multilinear extension (MLE) of a function f. Let l be a positive integer and $f : \{0,1\}^l \to \mathbb{F}$ be a function. The MLE of f, denoted by $\tilde{f}(\boldsymbol{X}) = \mathsf{MLE}(f)$, is the polynomial

$$\tilde{f}(\boldsymbol{X}) = \mathsf{MLE}(f) = \sum_{\mathbf{i} \in \{0,1\}^l} f(\mathbf{i}) \cdot \widetilde{\mathsf{eq}}_l(\boldsymbol{X} \| \mathbf{i}) \in \mathbb{F}[l],$$

where, for $\boldsymbol{X} = (X_1, \ldots, X_l)$ and $\mathbf{e} = (e_1, \ldots, e_l) \in \mathbb{F}^l$,

$$\widetilde{\mathsf{eq}}_l(\boldsymbol{X} \| \mathbf{e}) = \mathit{\Pi}_{i \in [l]}(X_i e_i + (1 - X_i)(1 - e_i)).$$

2.4 Polynomial Commitment Schemes (PCSs)

A polynomial commitment scheme (PCS, [15]) is a tuple $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$**:** On input 1^λ, ouput the public parameter pp.

$\mathsf{Com}(\mathsf{pp}, f(\boldsymbol{X})) \to (\sigma(f), \mathsf{aux})$**:** On input pp and $f(\boldsymbol{X}) \in \mathbb{F}[l]$ for some $l \in \mathbb{Z}_+$, output the polynomial commitment $\sigma(f)$ and an auxiliary information aux.

$\mathsf{Open}(\mathsf{pp}, \sigma(f), f(\boldsymbol{X}), \mathsf{aux}) \to b$**:** On input pp, polynomial commitment $\sigma(f)$, polynomial $f(\boldsymbol{X}) \in \mathbb{F}[l]$, for some $l \in \mathbb{Z}_+$, and an auxiliary input aux, return a bit $b \in \{0,1\}$ indicating accepted or rejected.

$\mathsf{Eval}\ \langle \mathcal{P}(f(\boldsymbol{X}), \mathsf{aux}), \mathcal{V} \rangle\ (\mathsf{pp}, \sigma(f), \mathbf{r}, e) \to b$**:** This is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with common inputs pp, $\sigma(f)$, $\mathbf{r} \in \{0,1\}^l$, for some $l \in \mathbb{Z}_+$, and $e \in \mathbb{F}$. $\mathcal{P}$ additionally holds $f(\boldsymbol{X}) \in \mathbb{F}[l]$ and aux. After the execution, $\mathcal{V}$ outputs a bit $b \in \{0,1\}$ indicating whether $\mathcal{V}$ accepts $f(\mathbf{r}) = e$.

Security. We briefly recall the security of a PCS. In Appendix B.2, we provide detailed security notions. A PCS is secure if it satisfies completeness and binding. A PCS is extractable if Eval is knowledge-sound. For *completeness*, if $\sigma(f)$ is correctly computed and $f(\mathbf{r}) = e$ are correct, $\mathcal{V}$ always accepts. Regarding *binding*, given a polynomial f and its commitment $\sigma(f)$, a PPT adversary

$\mathcal{A}$ cannot find a second opening f^*. Regarding *knowledge soundness*, given the commitment $\sigma(f)$, if a malicious prover $\mathcal{P}^*$ is accepted by $\mathcal{V}$ in Eval, one can extract the polynomial f by some extractor.

Notations. We write $\sigma(f)$ to indicate a polynomial commitment to $f(\boldsymbol{X}) \in \mathbb{F}[l]$, for some $l \in \mathbb{Z}_+$, when $f(\boldsymbol{X})$ is clear in the context. Moreover, regarding efficiency-related notations for PCSs, let $\mathsf{tp}(l)$, $\mathsf{tv}(l)$, $\mathsf{cs}(l)$ and $\mathsf{ps}(l)$ respectively denote the prover time (including committing and evaluating), verifier time, commitment size and the proof size from executing the evaluation protocol Eval from PCS for multilinear l-variate polynomials. Moreover, $\epsilon_{\mathsf{eval}}(l)$ denotes the soundness error of the evaluation of the employed PCS.

There are many PCSs [25,29,27] in the literature. Here, we focus on Sona (recalled in Table 1), a recent PCS introduced in [22].

Table 1. For $l \in \mathbb{Z}_+$, this table contains the costs of Sona [22] for committing and evaluating $f(\boldsymbol{X}) \in \mathbb{F}[l]$. $\mathbb{H}$ is collision-resistance hash, $\mathbb{F}$ is a finite field and $\mathbb{G}$ is cryptographic group. For commitment size, writing $1\mathbb{H}$ means that the commitment size is 1 hash value. For running time, writing $\mathcal{O}(2^l)\mathbb{F}$ means that it requires $\mathcal{O}(2^l)$ computations with operations in $\mathbb{F}$.

PCS	**Setup**	$\mathsf{cs}(l)$	$\mathsf{ps}(l)$	$\epsilon_{\mathsf{eval}}(l)$	$\mathsf{tp}(l)$	$\mathsf{tv}(l)$
Sona [22]	transparent	$1\mathbb{H}$	$\mathcal{O}(1)\mathbb{G}$	$\mathsf{negl}(l)$	$\mathcal{O}(2^l)\mathbb{F}, \mathcal{O}(\sqrt{2^l})\mathbb{G}$	$\mathcal{O}(\sqrt{2^l})\mathbb{G}$

2.5 Sumcheck Protocol and Lookup Protocol from Sumcheck

Sumcheck Protocol. The sumcheck protocol originally proposed in [17] is heavily studied to achieve SNARKs [21,6,22]. In a sumcheck protocol, given a polynomial $f(\boldsymbol{X}) \in \mathbb{F}[l]$ of degree at most ρ, for a claimed $S \in \mathbb{F}$, a prover $\mathcal{P}$ aims at proving $S = \sum_{i_1 \in \{0,1\}} \cdots \sum_{i_l \in \{0,1\}} f(i_1, \cdots, i_l)$. Instead of sending $f(\boldsymbol{X})$ directly, $\mathcal{P}$ executes an l-round interaction with the verifier $\mathcal{V}$. Let $\mathbf{r} = (r_1, \ldots, r_l)$ be the randomness that $\mathcal{V}$ chooses in the protocol. At the last step, $\mathcal{P}$ outputs $f_l(X_l)$ and $\mathcal{V}$ accepts if $f(\mathbf{r}) = e$ where $e = f_l(r_l)$.

When being instantiated to construct SNARKs, f is committed under some PCS to achieve commitment $\sigma(f)$. Then, the sumcheck $f(\mathbf{r}) = e$ is conducted through the evaluation protocol of the PCS. In Appendix B.3, we recall the sumcheck protocol Π_{sum} for relation

$$\mathcal{R}_{\mathsf{sum}} = \left\{ (S \in \mathbb{F}, \sigma(f);\ f(\boldsymbol{X}) \in \mathbb{F}[l], \mathsf{aux}) : S = \sum_{\mathbf{i} \in \{0,1\}^l} f(\mathbf{i}) \right\}, \tag{5}$$

where aux is the auxiliary input in the PCS. As discussed previously [17,21,22], the sumcheck protocol Π_{sum} satisfies completeness and knowledge soundness. The soundness error and efficiency of Π_{sum} are summarized in Table 2.

Lookup Argument Adapted from [14]. We consider the lookup protocol Π_{lkup} adapted from [14] for relation

$$\mathcal{R}_{\mathsf{lkup}}=\Big\{\big(\sigma(\tilde{f}_a),\sigma(\tilde{f}_b);\tilde{f}_a(\boldsymbol{X}),\tilde{f}_b(\boldsymbol{X}),\mathsf{aux}\big):\{\tilde{f}_a(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{l_a}}\subseteq\{\tilde{f}_b(\mathbf{i})\}_{\mathbf{i}\in\{0,1\}^{l_b}}\Big\} \quad (6)$$

where $l_a, l_b \in \mathbb{Z}_+$, $\tilde{f}_a(\boldsymbol{X}) \in \mathbb{F}[l_a]$ and $\tilde{f}_b(\boldsymbol{X}) \in \mathbb{F}[l_b]$, and aux is the auxiliary input in the PCS. The protocol Π_{lkup} shows the satisfaction of $\mathcal{R}_{\mathsf{lkup}}$ by reducing the lookup argument to sumcheck. Detailed protocol is deferred to Appendix B.3. See Table 2 for the related costs.

Remark 1 (Tuple Lookup). We will encounter tuple lookup arguments in this result. Here, tuple lookup captures the fact that a set of fixed-length tuples is a multisubset of another set of fixed-length tuples.

Specifically, let $\gamma, l_a, l_b \in \mathbb{Z}_+$, $n_a = 2^{l_a}$ and $n_b = 2^{l_b}$. Let $(\mathbf{a}_j)_{j\in[n_a]} \in (\mathbb{F}^\gamma)^{n_a}$ and $(\mathbf{b}_i)_{i\in[n_b]} \in (\mathbb{F}^\gamma)^{n_b}$. We would like to prove tuple lookup argument $\{\mathbf{a}_j\}_{j\in[n_a]} \subseteq \{\mathbf{b}_i\}_{i\in[n_b]}$. Notice that Π_{lkup} is only suitable for the cases that $\gamma = 1$, which we call the ordinary lookup arguments. Therefore, to conduct this tuple lookup argument, we use a challenge $\beta \xleftarrow{\$} \mathbb{F}$ and instead run Π_{lkup} to prove the reduced lookup argument $\left\{\left\langle \mathbf{a}_j, (\beta^{\hat{i}-1})_{\hat{i}\in[\gamma]}\right\rangle\right\}_{j\in[n_a]} \subseteq \left\{\left\langle \mathbf{b}_i, (\beta^{\hat{i}-1})_{\hat{i}\in[\gamma]}\right\rangle\right\}_{i\in[n_b]}$ which is an ordinary lookup. If the tuple lookup argument satisfies the lookup condition, then it is obvious that the reduced lookup argument is also satisfied.

However, regarding soundness, this reduction certainly incurs some soundness error which can be analyzed as follows. Assume that there exists some $j \in [n_a]$ s.t. $\mathbf{a}_j \notin \{\mathbf{b}_i\}_{i\in[n_b]}$. Then, for each i, the probability that $\left\langle \mathbf{a}_j, (\beta^{\hat{i}-1})_{\hat{i}\in[\gamma]}\right\rangle = \left\langle \mathbf{b}_i, (\beta^{\hat{i}-1})_{\hat{i}\in[\gamma]}\right\rangle$ is at most $= \mathcal{O}(\gamma/|\mathbb{F}|)$ by Schwartz-Zippel Lemma. Taking union bound over all $j \in [n_a]$ and all $i \in [n_b]$, the reduction error for the reduced lookup is at most $(\gamma-1)\cdot n_a \cdot n_b/|\mathbb{F}| = \mathcal{O}(\gamma \cdot n_a \cdot n_b/|\mathbb{F}|)$. Running Π_{lkup} (adapted from [14]) on this reduced lookup argument, we achieve the total soundness error $\mathcal{O}(\gamma \cdot n_a \cdot n_b/|\mathbb{F}| + \epsilon_{\mathsf{eval}}(l_a) + \epsilon_{\mathsf{eval}}(l_b))$ where ϵ_{eval} is introduced in Section 2.4.

Table 2. This table contains efficiency of protocols from sumcheck (c.f. [6,14]). Parameters l, l_a ad l_b are described in those protocols (see relations $\mathcal{R}_{\mathsf{sum}}$ and $\mathcal{R}_{\mathsf{lkup}}$ in (5) and (6), respectively). Notations $\mathsf{cs}, \mathsf{ps}, \mathsf{tp}, \mathsf{tv}$ and ϵ_{eval} are introduced in Section 2.4.

Protocol	Proof size	Soundness error	Prover time	Verifier time
Π_{sum} [6]	$\mathcal{O}(l\cdot\rho+\mathsf{ps}(l))$	$\mathcal{O}\left(\frac{l\cdot\rho}{\lvert\mathbb{F}\rvert}+\epsilon_{\mathsf{eval}}(l)\right)$	$\mathcal{O}(2^l\rho\log_2^2\rho+\mathsf{tp}(l))$	$\mathcal{O}(l\cdot\rho+\mathsf{tv}(l))$
Π_{lkup} [14]	$\mathcal{O}(l_a+l_b$ $+\mathsf{ps}(l_a)+\mathsf{ps}(l_b))$	$\mathcal{O}((l_a+l_b)/\lvert\mathbb{F}\rvert$ $+\epsilon_{\mathsf{eval}}(l_a)+\epsilon_{\mathsf{eval}}(l_b))$	$\mathcal{O}(2^{l_a}+2^{l_b}$ $+\mathsf{tp}(l_a)+\mathsf{tp}(l_b))$	$\mathcal{O}(l_a+l_b$ $+\mathsf{tv}(l_a)+\mathsf{tv}(l_b))$

3 Handling (Non-)Subsequence Arguments

In this section, we introduce our method to prove (non-)subsequence relations.

We now recall the initial discussion for proving non-subsequence arguments in Section 1.2 with the employment of algorithm SubseqSearch in Figure 1. In this setting, for $n, N \in \mathbb{Z}_+$ and alphabet $\Sigma \subseteq \mathbb{F}$, we have $\mathbf{s} \in \Sigma^n$, $\mathbf{t} \in \Sigma^N$ and sequence $(p_j)_{j\in[0,N]}$ viewed as the trace after executing SubseqSearch. Assume that $\mathbf{s}_\perp = (\mathbf{s} \| \perp)$, namely, a concatenation of $\mathbf{s}$ and some dummy character $\perp$ not belonging to Σ. Hence, we assume that $\mathbf{s}_\perp = (s_1, \ldots, s_n, s_{n+1})$ where $s_{n+1} = \perp$. This setting of s_{n+1} is tricky because it ensures $p_j \in [0, n]$ for all $j \in [N]$ since $\perp$ is different to all characters in $\mathbf{t}$. We also denote by $\Sigma_\perp = \Sigma \cup \{\perp\} = \{i\}_{i\in[0,|\Sigma|]}$, namely, a set of values from 0 to $|\Sigma|$. For simplicity, we assume that $\perp = 0$ and $\Sigma = \{1, \ldots, |\Sigma|\}$. We denote by $\mathbf{p} = (p_j)_{j\in[N]}$ which excludes $p_0 = 0$. In Section 1.2, to show that $\mathbf{p}$ is well-formed, we additionally use $\mathbf{v} = (v_j)_{j\in[N]}$ and show that $\mathbf{p}$ is well-formed, i.e., p_j is the largest index such that $(s_1, \ldots, s_{p_j}) \lhd (t_1, \ldots, t_j)$. This is formally proven in the following Lemma 2. We remark that system (7) is the aggregation of (2), (3) and (4).

Lemma 2 (Well-Formedness of p). *Let $n, N \in \mathbb{Z}_+$ and $\Sigma = \{i\}_{i\in[|\Sigma|]}$. Let $\mathbf{s} \in \Sigma^n$ and $\mathbf{t} \in \Sigma^N$. Let $\mathbf{p} = (p_j)_{j\in[N]} \in \mathbb{Z}_+^N$. Then, for $j \in [N]$, p_j is the largest index satisfying $(s_1, \ldots, s_{p_j}) \lhd (t_1, \ldots, t_j)$ iff there exists $\mathbf{v} = (v_j)_{j\in[N]}$ satisfying*

$$\begin{cases} \{(p_{j-1} + 1, v_j)\}_{j\in[N]} \subseteq \{(j, s_j)\}_{j\in[n+1]}, \\ \{(p_j - p_{j-1}, t_j - v_j)\}_{j\in[N]} \subseteq \{(0, i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1, 0)\}. \end{cases} \tag{7}$$

Proof. We first observe that

$$\{(p_{j-1} + 1, v_j)\}_{j\in[N]} \in \{(j, s_j)\}_{j\in[n+1]} \iff s_{p_{j-1}+1} = v_j \ \forall j \in [N].$$

On the other hand, we see that, for each $j \in [N]$,

$$\begin{aligned} &(p_j - p_{j-1}, t_j - v_j) \in \{(0, i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1, 0)\} \\ &\iff (p_j - p_{j-1} = 1 \wedge t_j = v_j) \vee (p_j - p_{j-1} = 0 \wedge t_j \neq v_j). \end{aligned}$$

Then, one sees that $p_j - p_{j-1}$ indicates whether $t_j = v_j$ (i.e., $p_j - p_{j-1} = 1$) or not (i.e., $p_j - p_{j-1} = 0$). Hence, by combining the above arguments, one has

$$\begin{aligned} &(s_{p_{j-1}+1} = v_j) \wedge ((p_j - p_{j-1} = 1 \wedge t_j = v_j) \vee (p_j - p_{j-1} = 0 \wedge t_j \neq v_j)) \\ &\iff (p_j - p_{j-1} = 1 \wedge s_{p_{j-1}+1} = t_j) \vee (p_j - p_{j-1} = 0 \wedge s_{p_{j-1}+1} \neq t_j). \end{aligned}$$

Finally, we achieve the equivalence

$$(7) \iff (p_j - p_{j-1} = 1 \wedge s_{p_{j-1}+1} = t_j) \vee (p_j - p_{j-1} = 0 \wedge s_{p_{j-1}+1} \neq t_j).$$

Equivalently, the RHS of the above equivalence can be written as

$$p_j - p_{j-1} = 1 \iff s_{p_{j-1}+1} = t_j \text{ or } p_j - p_{j-1} = 0 \iff s_{p_{j-1}+1} \neq t_j. \tag{8}$$

By Lemma 1, the sequence $(p_j)_{j\in[N]}$ satisfying above property captures the fact that, for $j \in [N]$, p_j is the largest index satisfying $(s_1, \ldots, s_{p_j}) \lhd (t_1, \ldots, t_j)$. This concludes the proof. □

Putting All Together. With well-formedness of $\mathbf{p}$ captured by Lemma 2, we can show that $\mathbf{s} \lhd \mathbf{t}$ or $\mathbf{s} \not\lhd \mathbf{t}$ by showing either $p_N = n$ (for $\mathbf{s} \lhd \mathbf{t}$) or $p_N < n$ (for $\mathbf{s} \not\lhd \mathbf{t}$). This is true because, by its definition, p_N is the largest index satisfying $(s_1, \ldots, s_{p_N}) \lhd (t_1, \ldots, t_N)$ and $p_N = n$ means that $\mathbf{s} = (s_1, \ldots, s_{p_N})$ while $p_N < n$ means that $(s_1, \ldots, s_{p_N})$ is a proper prefix of $\mathbf{s}$. Hence, we unify everything into the following Theorem 2.

Theorem 2. *Let $n, N \in \mathbb{Z}_+$. Let $\mathbf{s}_\perp = (s_j)_{j\in[n+1]} \in \Sigma_\perp^{n+1}$, and $\mathbf{t} = (t_j)_{j\in[N]} \in \Sigma^N$. Then, $\mathbf{s} \not\lhd \mathbf{t}$ (respectively, $\mathbf{s} \lhd \mathbf{t}$) iff there exist $(p_j)_{j\in[0,N]}$, with $p_0 = 0$ by default, and $\mathbf{v} = (v_j)_{j\in[N]}$ satisfying*

$$\begin{cases} \{s_j\}_{j\in[n]} \subseteq \Sigma, \quad s_{n+1} = \perp, \quad \{t_j\}_{j\in[N]} \subseteq \Sigma, \\ \{(p_{j-1}+1, v_j)\}_{j\in[N]} \subseteq \{(j, s_j)\}_{j\in[n+1]}, \\ \{(p_j - p_{j-1}, t_j - v_j)\}_{j\in[N]} \subseteq \{(0,i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1,0)\}, \end{cases} \tag{9}$$

and $p_N < n$ (respectively, $p_N = n$).

Proof. The first line of (9) captures the correct domains of $\mathbf{s}_\perp$ and $\mathbf{t}$. The remaining lines are from Lemma 2. Thus, the proof directly follows Lemma 2. □

4 Description of Succinct Non-Subsequence Arguments

In this section, we provide a protocol for non-subsequence arguments from sumcheck. We first introduce frequently used notations in Section 4.1. In Section 4.2, we discuss the transformations of constraints in Theorem 2 into the forms suitable for using sumcheck. The description and efficiency of our non-subsequence argument are in Sections 4.3 and 4.4, respectively.

4.1 Notations

We recall from Section 2.1 the encoding $\mathsf{int} : \{0,1\}^l \to \mathbb{Z}_+$, for some $l \in \mathbb{Z}_+$, that maps any vector $\mathbf{i} = (i_1, \ldots, i_l) \in \{0,1\}^l$ into $i = \sum_{j\in[l]} 2^{j-1} \cdot i_j$. This notation is suitable for 0-based indexing. However, in some places of our result, we may use 1-based indexing instead. Therefore, we additionally introduce

$$\mathsf{inc}_l : \{0,1\}^l \to \mathbb{Z}_+ \text{ mapping } \mathbf{i} \mapsto \mathsf{int}_l(\mathbf{i}) + 1$$

where inc stands for "increment". We denote by $\widetilde{\mathsf{int}_l}(\boldsymbol{X}) \in \mathbb{F}[l]$ and $\widetilde{\mathsf{inc}_l}(\boldsymbol{X}) \in \mathbb{F}[l]$ respectively the multilinear extensions of int_l and inc_l. Define

$$l_N = \log_2 N, \; l_\Sigma \text{ s.t. } 2^{l_\Sigma} \geq |\Sigma_\perp|, l_n = \log_2(n+1) \in \mathbb{Z}_+.$$

We now define the following functions encoding components $\mathbf{s}_\perp = (s_j)_{j\in[n+1]}$, $\mathbf{t} = (t_j)_{j\in[N]}$, $\mathbf{p} = (p_j)_{j\in[N]}$, $\mathbf{d} = (d_j)_{j\in[N]} = (p_{j-1})_{j\in[N]}$ and $\mathbf{v} = (v_j)_{j\in[N]}$ introduced in Theorem 2.

$$
\begin{aligned}
&f_s : \{0,1\}^{l_n} \to \mathbb{F} \text{ maps } \mathbf{j} \mapsto s_j && \forall \mathbf{j} \in \{0,1\}^{l_n} \text{ where } j = \widetilde{\mathsf{inc}}_{l_n}(\mathbf{j}).\\
&f_t : \{0,1\}^{l_N} \to \mathbb{F} \text{ maps } \mathbf{j} \mapsto t_j && \forall \mathbf{j} \in \{0,1\}^{l_N} \text{ where } j = \widetilde{\mathsf{inc}}_{l_N}(\mathbf{j}).\\
&f_p : \{0,1\}^{l_N} \to \mathbb{F} \text{ maps } \mathbf{j} \mapsto p_j && \forall \mathbf{j} \in \{0,1\}^{l_N} \text{ where } j = \widetilde{\mathsf{inc}}_{l_N}(\mathbf{j}).\\
&f_d : \{0,1\}^{l_N} \to \mathbb{F} \text{ maps } \mathbf{j} \mapsto d_j = p_{j-1} && \forall \mathbf{j} \in \{0,1\}^{l_N} \text{ where } j = \widetilde{\mathsf{inc}}_{l_N}(\mathbf{j}).\\
&f_v : \{0,1\}^{l_N} \to \mathbb{F} \text{ maps } \mathbf{j} \mapsto v_j && \forall \mathbf{j} \in \{0,1\}^{l_N} \text{ where } j = \widetilde{\mathsf{inc}}_{l_N}(\mathbf{j}).
\end{aligned}
$$

Notice here that f_d is introduced to encode $\mathbf{d} = (d_j)_{j\in[N]} = (p_{j-1})_{j\in[N]}$. This is due to the constraint $\{(p_j - p_{j-1}, t_j - v_j)\}_{j\in[N]} \subseteq \{(0,i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1,0)\}$ in Theorem 2 requiring computing $p_j - p_{j-1}$, for all $j \in [N]$, which is a linear combination of elements in entries of $(p_j)_{j\in[0,N]}$. Therefore, we extract $(p_{j-1})_{j\in[N]}$ and encode them into f_d for ease of handling such constraint.

Denote by $\tilde{f}_s(\boldsymbol{X}) \in \mathbb{F}[l_n]$, $\tilde{f}_t(\boldsymbol{X}) \in \mathbb{F}[l_N]$, $\tilde{f}_p(\boldsymbol{X}) \in \mathbb{F}[l_N]$, $\tilde{f}_d(\boldsymbol{X}) \in \mathbb{F}[l_N]$ and $\tilde{f}_v(\boldsymbol{X}) \in \mathbb{F}[l_N]$ to be multilinear extensions of above functions. We also assume that prover commits to $\tilde{f}_s(\boldsymbol{X})$, $\tilde{f}_t(\boldsymbol{X})$, $\tilde{f}_p(\boldsymbol{X})$, $\tilde{f}_d(\boldsymbol{X})$ and $\tilde{f}_v(\boldsymbol{X})$ to obtain polynomial commitments $\sigma(\tilde{f}_s)$, $\sigma(\tilde{f}_t)$, $\sigma(\tilde{f}_p)$, $\sigma(\tilde{f}_d)$ and $\sigma(\tilde{f}_v)$, respectively.

4.2 Transformations

We aim to design a protocol from sumcheck for relation

$$
\mathcal{R}_{\mathsf{non\text{-}subseq}} = \left\{ \left(\sigma(\tilde{f}_s), \sigma(\tilde{f}_t);\ \tilde{f}_s(\boldsymbol{X}), \tilde{f}_t(\boldsymbol{X}), \mathsf{aux}\right) \colon \mathbf{s} \not\lhd \mathbf{t} \right\}
$$

where $\tilde{f}_s(\boldsymbol{X})$ and $\tilde{f}_t(\boldsymbol{X})$ encode $\mathbf{s}$ and $\mathbf{t}$, respectively, as introduced in Section 4.1, and aux is some auxiliary input for executing evaluation protocol of PCS. Our idea of proving $\mathbf{s} \not\lhd \mathbf{t}$ strictly follows system (9) in Theorem 2. We divide the constraints in (9) into following cases w.r.t. the notations in Section 4.1:

Proving $\{s_j\}_{j\in[n]} \subseteq \Sigma$, $s_{n+1} = \perp$ and $\{t_j\}_j \subseteq \Sigma$. We first encode $\Sigma_\perp$ into a polynomial as follows. Let $l_\Sigma \in \mathbb{Z}_+$ satisfy $2^{l_\Sigma} \geq |\Sigma_\perp|$. We form a sequence $\mathsf{char} = (\mathsf{char}_i)_{i\in[2^{l_\Sigma}]}$ from $\Sigma_\perp$ s.t. $\{\mathsf{char}_i\}_{i\in[2^{l_\Sigma}]} = \Sigma_\perp$. Notice that $|\Sigma_\perp|$ may not be a power of 2. Therefore, we simply duplicate elements (if necessary) to reach the desired cardinality. Hence, $\{s_j\}_{j\in[n+1]} \subseteq \{\mathsf{char}_i\}_{i\in[2^{l_\Sigma}]}$ iff $\{s_j\}_{j\in[n+1]} \subseteq \Sigma_\perp$. Define $g_\Sigma : \{0,1\}^{l_\Sigma} \to \mathbb{F}$ that maps $\mathbf{i} \mapsto \mathsf{char}_i$ for $i = \mathsf{inc}_{l_\Sigma}(\mathbf{i})$. To show that $\{s_j\}_{j\in[n]} \subseteq \Sigma$ and $s_{n+1} = \perp$, define sequences $\mathsf{ch} = (\mathsf{ch}_i)_{i\in[2^{l_\Sigma}]} \in \{0,1\}^{2^{l_\Sigma}}$ and $\mathsf{ch}' = (\mathsf{ch}'_j)_{j\in[n+1]} \in \{0,1\}^{n+1}$ s.t.

$$
\mathsf{ch}_i = 1 \iff \mathsf{char}_i \neq \perp \ \ \forall i \in [2^{l_\Sigma}] \text{ and } \mathsf{ch}'_j = 1 \iff s_j \neq \perp \ \ \forall j \in [n+1].
$$

Notice here that both ch and ch' can be determined by any party. We see that

$$
\{s_j\}_{j\in[n]} \subset \Sigma \wedge s_{n+1} = \perp \iff \{(\mathsf{ch}'_j, s_j)\}_{j\in[n+1]} \subseteq \{(\mathsf{ch}_i, \mathsf{char}_i)\}_{i\in[2^{l_\Sigma}]}. \quad (10)
$$

Since we would like to show the LHS of (10), we equivalently show that its RHS holds. Therefore, we encode ch and ch' by $\tilde{g}_{\mathsf{ch}}(\boldsymbol{X}) \in \mathbb{F}[l_\Sigma]$, mapping $\mathbf{i} \mapsto \mathsf{ch}_i$

for $i = \mathsf{inc}_{l_\Sigma}(\mathbf{i})$, and $\tilde{g}_{\mathsf{ch}'}(\boldsymbol{X}) \in \mathbb{F}[l_n]$, mapping $\mathbf{j} \mapsto \mathsf{ch}'_j$ for $j = \mathsf{inc}_{l_N}(\mathbf{j})$. Lookup argument $\{(\mathsf{ch}'_j, s_j)\}_{j\in[n+1]} \subseteq \{(\mathsf{ch}_i, \mathsf{char}_i)\}_{i\in[2^{l_\Sigma}]}$ in RHS of (10) is hence proved through reduced lookup $\{\widetilde{\mathsf{vs}}(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{l_n}} \subseteq \{\tilde{h}_\Sigma(\mathbf{i})\}_{\mathbf{i}\in\{0,1\}^{l_\Sigma}}$, where

$$\tilde{h}_\Sigma(\boldsymbol{X}) = \tilde{g}_{\mathsf{ch}}(\boldsymbol{X}) + \tilde{g}_\Sigma(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_\Sigma], \quad \widetilde{\mathsf{vs}}(\boldsymbol{X}) = \tilde{g}_{\mathsf{ch}'}(\boldsymbol{X}) + \tilde{f}_s(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_n] \tag{11}$$

and $\beta \xleftarrow{\$} \mathbb{F}^\star$, with reduction error at most $2^{l_n} \cdot 2^{l_\Sigma} / |\mathbb{F}| = \mathcal{O}(n \cdot |\Sigma| \,/\, |\mathbb{F}|)$ (see Remark 1). Similarly, we can show that $\{\widetilde{\mathsf{vt}}(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{l_N}} \subseteq \{\tilde{h}_\Sigma(\mathbf{i})\}_{\mathbf{i}\in\{0,1\}^{l_\Sigma}}$, where

$$\widetilde{\mathsf{vt}}(\boldsymbol{X}) = 1 + \tilde{f}_t(\boldsymbol{X}) \cdot \beta, \tag{12}$$

to prove $\{t_j\}_{j\in[N]} \subseteq \Sigma$ with reduction error $\mathcal{O}\left(N \cdot |\Sigma| \,/\, |\mathbb{F}|\right)$.

Well-Formedness of $\tilde{f}_d(\boldsymbol{X}) \in \mathbb{F}[l_N]$. In Section 4.1, $\tilde{f}_d(\boldsymbol{X}) \in \mathbb{F}[l_N]$ encodes $\mathbf{d} = (d_j)_{j\in[N]} = (p_{j-1})_{j\in[N]}$. As $\tilde{f}_p(\boldsymbol{X}) \in \mathbb{F}[l_N]$ encodes $\mathbf{p} = (p_j)_{j\in[N]}$, we see that $\tilde{f}_d(\boldsymbol{X})$ has a tight relationship with $\tilde{f}_p(\boldsymbol{X})$. In fact, this is a linear mapping from $\mathbf{p}$ to $\mathbf{d}$ written as $\mathbf{d} = \mathbf{A} \cdot \mathbf{p}$ for some $\mathbf{A} \in \mathbb{F}^{N\times N}$ of the form

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \in \mathbb{F}^{N\times N} \text{ with exactly } N-1 \text{ non-zero entries.}$$

To prove $\mathbf{d} = \mathbf{A} \cdot \mathbf{p}$, we follow [21] to encode $\mathbf{A}$ into $\tilde{g}_A(\boldsymbol{Y} \| \boldsymbol{X}) \in \mathbb{F}[2l_N]$ and commit to it by some PCS. Hence, we show well-formedness of $\mathbf{d}$ by proving

$$\tilde{f}_d(\boldsymbol{Y}) = \sum_{\mathbf{j}\in\{0,1\}^{l_N}} \tilde{g}_A(\boldsymbol{Y} \| \mathbf{j}) \cdot \tilde{f}_p(\mathbf{j}). \tag{13}$$

A way to prove this equation is to have $\mathbf{r}_d \xleftarrow{\$} \mathbb{F}^{l_N}$ from the verifier. Then, prover shows that $e_d = \tilde{f}_d(\mathbf{r}_d) = \sum_{\mathbf{j}\in\{0,1\}^{l_N}} \tilde{g}_A(\mathbf{r}_d \| \mathbf{j}) \cdot \tilde{f}_p(\mathbf{j})$. In this way, the prover must commit to $\tilde{g}_r(\boldsymbol{X}) = \tilde{g}_A(\mathbf{r}_d \| \boldsymbol{X}) \in \mathbb{F}[l_N]$ to make succinct proof size and sublinear verification time. Notice that $\mathbf{A}$ is sparse, i.e., having exactly $N-1$ non-zero entries. Therefore, we can write

$$\begin{aligned} \tilde{g}_A(\boldsymbol{Y} \| \boldsymbol{X}) &= \sum_{i,j\in[N]} a_{i,j} \cdot \widetilde{\mathsf{eq}}_{l_N}(\boldsymbol{Y} \| \mathsf{bin}_{l_N}(i)) \cdot \widetilde{\mathsf{eq}}_{l_N}(\boldsymbol{X} \| \mathsf{bin}_{l_N}(j)) \\ &= \sum_{i,j\in[N] \text{ s.t. } a_{i,j}\neq 0} a_{i,j} \cdot \widetilde{\mathsf{eq}}_{l_N}(\boldsymbol{Y} \| \mathsf{bin}_{l_N}(i)) \cdot \widetilde{\mathsf{eq}}_{l_N}(\boldsymbol{X} \| \mathsf{bin}_{l_N}(j)). \end{aligned} \tag{14}$$

Hence, to evaluate $\tilde{g}_A(\mathbf{r}_d \| \boldsymbol{X})$, one can evaluate all $a'_{i,j} = a_{i,j} \cdot \widetilde{\mathsf{eq}}_{l_N}(\mathbf{r}_d \| \mathsf{bin}(i))$ for $i, j \in [N]$ satisfying $a_{i,j} \neq 0$. This cost $\mathcal{O}(N)$ time for prover as prover can compute $\widetilde{\mathsf{eq}}_{l_N}(\mathbf{r}_d \| \mathbf{i})$ for all $\mathbf{i} \in \{0,1\}^{l_N}$ as follows. Let $\mathbf{r}_d = (r_{d,1}, \ldots, r_{d,l_N})$ and $\mathbf{i} = (i_1, \ldots, i_{l_N})$. Then, by definition, we have

$$\widetilde{\mathsf{eq}}_{l_N}(\mathbf{r}_d \| \mathbf{i}) = \prod_{j\in[l_N]} (r_{d,j} \cdot i_j + (1 - r_{d,j}) \cdot (1 - i_j)).$$

Set $P_0 := 1$.
For each k form 1 to l_N, the k-th round runs as follows:
- For all $(i_1, \ldots, i_k) \in \{0,1\}^k$:
  - Compute $P_{k,(i_1,\ldots,i_k)} := P_{k-1,(i_1,\ldots,i_{k-1})} \cdot (r_{d,k} \cdot i_k + (1 - r_{d,k}) \cdot (1 - i_k))$.

Fig. 2. Algorithm for computing all $\widetilde{\mathsf{eq}}(\mathbf{r}_d \| \mathbf{i})$ for all $\mathbf{i} \in \{0,1\}^{l_N}$.

By denoting $P_{k,(i_1,\ldots,i_k)} := \prod_{j\in[k]}(r_{d,j} \cdot i_j + (1 - r_{d,j}) \cdot (1 - i_j))$ for $k \in [l_N]$ and $(i_1, \ldots, i_k) \in \{0,1\}^k$, we see that

$$P_{k,(i_1,\ldots,i_{k-1},i_k)} = P_{k-1,(i_1,\ldots,i_{k-1})} \cdot (r_{d,k} \cdot i_k + (1 - r_{d,k}) \cdot (1 - i_k)).$$

With this idea, we can compute $\widetilde{\mathsf{eq}}_{l_N}(\mathbf{r}_d \| \mathbf{i})$ for all $\mathbf{i} \in \{0,1\}^{l_N}$ in time $\mathcal{O}(N)$ by using algorithm in Figure 2. This algorithm in the k-th round has time complexity $\mathcal{O}(2^k)$. Hence, in all l_N rounds, it costs total time $\sum_{k\in[l_N]} \mathcal{O}(2^k) = \mathcal{O}(2^{l_N})$. Prover now can compute $a'_{i,j} = a_{i,j} \cdot \widetilde{\mathsf{eq}}_{l_N}(\mathbf{r}_d \| \mathsf{bin}(i))$ for all $i, j \in [N]$ satisfying $a_{i,j} \neq 0$ in time $\mathcal{O}(N)$ by using $P_{l_N,\mathbf{i}} = \widetilde{\mathsf{eq}}_{l_N}(\mathbf{r}_d \| \mathbf{i})$ where $\mathbf{i} = \mathsf{bin}_{l_N}(i)$.

Finally, prover commit to

$$\tilde{g}_r(\boldsymbol{X}) = \tilde{g}_A(\mathbf{r}_d \| \boldsymbol{X}) = \sum_{i,j\in[N] \text{ s.t. } a_{i,j}\neq 0} a'_{i,j} \cdot \widetilde{\mathsf{eq}}_{l_N}(\boldsymbol{X} \| \mathsf{bin}_{l_N}(j)).$$

However, this way will incur verifier time to be $\mathcal{O}(N)$ to recompute $\tilde{g}_A(\mathbf{r}_d \| \boldsymbol{X})$ since this is the only way for verifier to check whether prover compute commitment to $\tilde{g}_A(\mathbf{r}_d \| \boldsymbol{X})$ honestly after pre-precessing. To reduce verifier time, we use techniques for committing to sparse polynomials, i.e., l-variate polynomials having a few non-zero entries among evaluations at $\mathbf{i} \in \{0,1\}^l$ for some $l \in \mathbb{Z}_+$, from [21,22] to optimize prover and verifier time with properties specified in Table 3.

Table 3. This table contains properties of sparse PCS [21,22] for $(2l_N)$-variate polynomials with at most N non-zero entries. This sparse PCS is a generic construction employing a PCS as a building block. Notations $\mathsf{cs}, \mathsf{ps}, \mathsf{tp}, \mathsf{tv}$ and ϵ_{eval} (introduced in Section 2.4) are for PCS.

Commitment size	**Proof size**	**Soundness error**	**Prover time**	**Verifier time**
$\mathsf{cs}(l_N)$	$\mathcal{O}(\mathsf{ps}(l_N))$	$\mathcal{O}\left(\frac{N}{\lvert\mathbb{F}\rvert} + \epsilon_{\mathsf{eval}}(l_N)\right)$	$\mathcal{O}(\mathsf{tp}(l_N))$	$\mathcal{O}(\mathsf{tv}(l_N))$

Hence, in the execution, initially, we have prover to commit to $\tilde{g}_A(\boldsymbol{Y} \| \boldsymbol{X})$ by the sparse PCS and verifier to send $\mathbf{r}_d$. Prover computes $\tilde{g}_r(\boldsymbol{X}) = \tilde{g}_A(\mathbf{r}_d \| \boldsymbol{X})$ in time $\mathcal{O}(N)$. Let $e_d = \tilde{f}_d(\mathbf{r}_d)$. Prover and verifier follow Π_{sum} for reducing argument $e_d = \sum_{\mathbf{j}\in\{0,1\}^{l_N}} \tilde{g}_r(\mathbf{j}) \cdot \tilde{f}_p(\mathbf{j})$ to checking that $e_p = \tilde{g}_r(\mathbf{r}_p) \cdot \tilde{f}_p(\mathbf{r}_p)$ with $(\mathbf{r}_p, e_p)$ is output of Π_{sum}. Eventually, run evaluation of sparse PCS to evaluate $\tilde{g}_A(\boldsymbol{Y} \| \boldsymbol{X})$ and of PCS to evaluate $\tilde{g}_r(\boldsymbol{Y})$ and $\tilde{f}_p(\boldsymbol{X})$ to check whether $\tilde{g}_A(\mathbf{r}_d \| \mathbf{r}_p) = \tilde{g}_r(\mathbf{r}_p)$ (implying $\tilde{g}_A(\mathbf{r}_d \| \boldsymbol{X}) = \tilde{g}_r(\boldsymbol{X})$) and $e_p = \tilde{g}_r(\mathbf{r}_p) \cdot \tilde{f}_p(\mathbf{r}_p)$.

Proving $\{(p_{j-1}+1, v_j)\}_{j\in[N]} \subseteq \{(j, s_j)\}_{j\in[n+1]}$. As $\tilde{f}_d(\boldsymbol{X})$ encodes sequence $\mathbf{d} = (d_j)_{j\in[N]} = (p_{j-1})_{j\in[N]}$, we prove this argument by defining

$$\begin{aligned}\widetilde{\mathsf{vv}}(\boldsymbol{X}) &= (\tilde{f}_d(\boldsymbol{X}) + 1) + \tilde{f}_v(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_N],\\ \widetilde{\mathsf{vx}}(\boldsymbol{X}) &= \widetilde{\mathsf{inc}}_{l_n}(\boldsymbol{X}) + \tilde{f}_s(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_n]\end{aligned} \tag{15}$$

where β is the challenge discussed above in (11). Then, we can reduce proving $\{(p_{j-1}+1, v_j)\}_{j\in[N]} \subseteq \{(j, s_j)\}_{j\in[n+1]}$ to $\{\widetilde{\mathsf{vv}}(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{l_N}} \subseteq \{\widetilde{\mathsf{vx}}(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{l_n}}$ with reduction error at most $2^{l_n} \cdot 2^{l_N} / |\mathbb{F}| = \mathcal{O}(n \cdot N / |\mathbb{F}|)$ (see Remark 1).

Proving $\{(p_j - p_{j-1}, t_j - v_j)\}_{j\in[N]} \subseteq \{(0, i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1, 0)\}$. To show this tuple lookup argument, we introduce the sequence $(\mathsf{iz}_i, \mathsf{diff}_i)_{i\in[m]}$ for some $m \in \mathbb{Z}_+$ s.t. $\{(\mathsf{iz}_i, \mathsf{diff}_i)\}_{i\in[m]} = \{(0, i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1, 0)\}$. Here, for ease of handling with sumcheck arguments, we assume that m is a power of 2 and $m = \mathcal{O}(|\Sigma|)$. Hence, if $\{(0, i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1, 0)\}$ is less than m entries, we simply duplicate elements (if necessary) s.t. $\{(\mathsf{iz}_i, \mathsf{diff}_i)\}_{i\in[m]}$ has exactly m entries. Let $l_m = \log_2 m$. We now encode them as follows.

$$\begin{aligned}&g_{\mathsf{iz}} : \{0,1\}^{l_m} \to \mathbb{F} \text{ maps } \mathbf{i} \mapsto \mathsf{iz}_i \qquad \forall \mathbf{i} \in \{0,1\}^{l_m} \text{ where } i = \widetilde{\mathsf{inc}}_{l_m}(\mathbf{i}).\\ &g_{\mathsf{diff}} : \{0,1\}^{l_m} \to \mathbb{F} \text{ maps } \mathbf{i} \mapsto \mathsf{diff}_i \quad \forall \mathbf{i} \in \{0,1\}^{l_m} \text{ where } i = \widetilde{\mathsf{inc}}_{l_m}(\mathbf{i}).\end{aligned}$$

Denote by $\tilde{g}_{\mathsf{iz}}(\boldsymbol{X}) = \mathsf{MLE}(g_{\mathsf{iz}}) \in \mathbb{F}[l_m]$ and $\tilde{g}_{\mathsf{diff}}(\boldsymbol{X}) = \mathsf{MLE}(g_{\mathsf{diff}}) \in \mathbb{F}[l_m]$. We now observe that $(p_j - p_{j-1})_{j\in[N]}$ is encoded by $\tilde{f}_p(\boldsymbol{X}) - \tilde{f}_d(\boldsymbol{X})$. Therefore, for β introduced in (11), by defining

$$\begin{aligned}\widetilde{\mathsf{vc}}(\boldsymbol{X}) &= (\tilde{f}_p(\boldsymbol{X}) - \tilde{f}_d(\boldsymbol{X})) + (\tilde{f}_t(\boldsymbol{X}) - \tilde{f}_v(\boldsymbol{X})) \cdot \beta \in \mathbb{F}[l_N],\\ \tilde{h}_{\mathsf{diff}}(\boldsymbol{X}) &= \tilde{g}_{\mathsf{iz}}(\boldsymbol{X}) + \tilde{g}_{\mathsf{diff}}(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_m],\end{aligned} \tag{16}$$

we can reduce proving $\{(p_j - p_{j-1}, t_j - v_j)\}_{j\in[N]} \subseteq \{(0, i)\}_{i\in[-|\Sigma|,|\Sigma|]\setminus\{0\}} \cup \{(1, 0)\}$ to proving $\{\widetilde{\mathsf{vc}}(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{l_N}} \subseteq \{\tilde{h}_{\mathsf{diff}}(\mathbf{i})\}_{\mathbf{i}\in\{0,1\}^{l_m}}$ with reduction error at most $2^{l_N} \cdot 2^{l_m} / |\mathbb{F}| = \mathcal{O}(N \cdot |\Sigma| / |\mathbb{F}|)$ (see Remark 1) since $m = \mathcal{O}(|\Sigma|)$.

4.3 Description for Non-Subsequence Arguments from Sumcheck

We present our non-subsequence argument $\Pi_{\mathsf{non\text{-}subseq}}$ for relation $\mathcal{R}_{\mathsf{non\text{-}subseq}}$ in Figure 3 w.r.t. notations and transformations in Sections 4.1 and 4.2, respectively. The protocol is divided into two phases, namely, pre-processing and execution. The pre-processing phase is for committing supporting polynomials $\tilde{f}_p(\boldsymbol{X})$, $\tilde{f}_d(\boldsymbol{X})$ and $\tilde{f}_v(\boldsymbol{X})$ in advance. Regarding the execution phase, the prover and verifier proceed with the interactive proving as discussed above. The security of protocol $\Pi_{\mathsf{non\text{-}subseq}}$ is discussed in Theorem 3. Protocol $\Pi_{\mathsf{non\text{-}subseq}}$ can be transformed into a SNARK by applying Fiat-Shamir heuristics [8].

Theorem 3 (Security of $\Pi_{\mathsf{non\text{-}subseq}}$). *Let ϵ_i be the soundness error of Π_i for $i \in [5]$. $\Pi_{\mathsf{non\text{-}subseq}}$ is complete, with completeness error $\mathcal{O}((2^{l_n} + 2^{l_N} + 2^{l_\Sigma}) / |\mathbb{F}|)$,*

and knowledge-sound, with soundness error

$$\mathcal{O}\left(\frac{n\cdot N + n\cdot|\Sigma| + N\cdot|\Sigma|}{|\mathbb{F}|} + \sum_{i\in[5]}\epsilon_i\right),$$

where ϵ_{eval} *is introduced in Section 2.4, if (i)* $\Pi_1,\ldots,\Pi_5$ *are complete and knowledge-sound, and (ii) employed PCS is secure and extractable.*

In particular, when Π_{lkup} *and* Π_3 *are adapted from [14] and [21,22], respectively, by following Tables 2 and 3, we have*

$$\sum_{i\in[5]}\epsilon_i = \mathcal{O}\left((l_n + l_N + l_\Sigma)/|\mathbb{F}| + \epsilon_{\mathsf{eval}}(l_n) + \epsilon_{\mathsf{eval}}(l_N) + \epsilon_{\mathsf{eval}}(l_\Sigma)\right).$$

Proof. Completeness is straightforward. Completeness error is computed due to the union bound on completeness errors of Π_1, Π_2, Π_4, and Π_5. In other words, the completeness error is bounded by

$$\begin{aligned}&\mathcal{O}\left(\frac{2^{l_n}+2^{l_\Sigma}}{|\mathbb{F}|}\right) + \mathcal{O}\left(\frac{2^{l_N}+2^{l_\Sigma}}{|\mathbb{F}|}\right) + \mathcal{O}\left(\frac{2^{l_N}+2^{l_n}}{|\mathbb{F}|}\right) + \mathcal{O}\left(\frac{2^{l_N}+2^{l_m}}{|\mathbb{F}|}\right)\\ &= \mathcal{O}\left(\frac{2^{l_n}+2^{l_N}+2^{l_\Sigma}+2^{l_m}}{|\mathbb{F}|}\right) = \mathcal{O}\left(\frac{2^{l_n}+2^{l_N}+2^{l_\Sigma}}{|\mathbb{F}|}\right)\end{aligned}$$

since $\mathcal{O}(2^{l_m}) = \mathcal{O}(m) = \mathcal{O}(|\Sigma|) = \mathcal{O}(2^{l_\Sigma})$. Regarding knowledge soundness, we first notice that, for the evaluations on $\tilde{f}_s(\boldsymbol{X}), \tilde{f}_t(\boldsymbol{X}), \tilde{f}_p(\boldsymbol{X}), \tilde{f}_d(\boldsymbol{X}), \tilde{f}_v(\boldsymbol{X})$ and $\tilde{g}_r(\boldsymbol{X})$, both parties run the evaluation protocol at least once since running protocols $\Pi_1,\ldots,\Pi_5$ w.r.t. commitments $\sigma(\tilde{f}_s), \sigma(\tilde{f}_t), \sigma(\tilde{f}_p), \sigma(\tilde{f}_d), \sigma(\tilde{f}_v)$ and $\sigma(\tilde{g}_r)$. Since the evaluation process of extractable PCSs is knowledge sound, there are corresponding extractors to extract $\tilde{f}_s(\boldsymbol{X}), \tilde{f}_t(\boldsymbol{X}), \tilde{f}_p(\boldsymbol{X}), \tilde{f}_d(\boldsymbol{X}), \tilde{f}_v(\boldsymbol{X})$ and $\tilde{g}_r(\boldsymbol{X})$. Hence, we can obtain $\mathbf{s}, \mathbf{t}, \mathbf{p}, \mathbf{d}$ and $\mathbf{v}$ from those extracted polynomials as in step 3 of protocol $\Pi_{\mathsf{non\text{-}subseq}}$ in Figure 3. Moreover, from extracted and common polynomials, we can also form polynomials $\tilde{h}_\Sigma(\boldsymbol{X})$, $\tilde{f}_{\mathsf{diff}}(\boldsymbol{X})$, $\widetilde{\mathsf{vs}}(\boldsymbol{X})$, $\widetilde{\mathsf{vt}}(\boldsymbol{X})$, $\widetilde{\mathsf{vv}}(\boldsymbol{X})$, $\widetilde{\mathsf{vx}}(\boldsymbol{X})$ and $\widetilde{\mathsf{vc}}(\boldsymbol{X})$. Since $\Pi_1,\ldots\Pi_5$ are knowledge sound and are all accepted by $\mathcal{V}$, we hence deduce the followings:

- $\{\widetilde{\mathsf{vs}}(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{\iota_n}} \subseteq \{\tilde{h}_\Sigma(\mathbf{i})\}_{\mathbf{i}\in\{0,1\}^{\iota_\Sigma}}$, from Π_1, holds with error probability ϵ_1. Notice that $\tilde{h}_\Sigma(\boldsymbol{X}) = \tilde{g}_{\mathsf{ch}}(\boldsymbol{X}) + \tilde{g}_\Sigma(\boldsymbol{X})\cdot\beta$, $\widetilde{\mathsf{vs}}(\boldsymbol{X}) = \tilde{g}_{\mathsf{ch}'}(\boldsymbol{X}) + \tilde{f}_s(\boldsymbol{X})\cdot\beta$ and $\beta \xleftarrow{\$} \mathbb{F}^\star$. Since common polynomials $\tilde{g}_{\mathsf{ch}}(\boldsymbol{X})$ and $\tilde{g}_{\mathsf{ch}'}(\boldsymbol{X})$ encode $\mathsf{ch} = (\mathsf{ch}_i)_{i\in[2^{\iota_\Sigma}]}$ and $\mathsf{ch}' = (\mathsf{ch}'_j)_{j\in[n+1]}$, respectively, with constraints in (11), we deduce that RHS of (10) holds with error probability $\left(1 - \mathcal{O}\left(\frac{n\cdot|\Sigma|}{|\mathbb{F}|}\right)\right)\cdot\epsilon_1 + \mathcal{O}\left(\frac{n\cdot|\Sigma|}{|\mathbb{F}|}\right)$ which is simplified to be $\mathcal{O}\left(\epsilon_1 + \frac{n\cdot|\Sigma|}{|\mathbb{F}|}\right)$ where $\mathcal{O}\left(\frac{n\cdot|\Sigma|}{|\mathbb{F}|}\right)$ is soundness error of reducing tuple lookup $\{(\mathsf{ch}'_j, s_j)\}_{j\in[n+1]} \subseteq \{(\mathsf{ch}_i, a_i)\}_{i\in[2^{\iota_\Sigma}]}$ to $\{\widetilde{\mathsf{vs}}(\mathbf{j})\}_{\mathbf{j}\in\{0,1\}^{\iota_n}} \subseteq \{\tilde{h}_\Sigma(\mathbf{i})\}_{\mathbf{i}\in\{0,1\}^{\iota_\Sigma}}$ (see Section 4.2). RHS of (10) hence implies its LHS, namely, $\{s_j\}_{j\in[n]} \subset \Sigma \wedge s_{n+1} = \perp$.

– As from above, reducing to and proving $\{\widetilde{\mathsf{vt}}(\mathbf{j})\}_{\mathbf{j}} \subseteq \{\tilde{h}_{\Sigma}(\mathbf{i})\}_{\mathbf{i}}$ with Π_2 have soundness error $\mathcal{O}\left(\epsilon_2 + \frac{N \cdot |\Sigma|}{|\mathbb{F}|}\right)$.
– $e_d = \sum_{\mathbf{j} \in \{0,1\}^{l_N}} \tilde{g}_r(\mathbf{j}) \cdot \tilde{f}_p(\mathbf{j})$, from Π_3, and the check $\tilde{g}_r(\mathbf{r}_p) = \tilde{g}_A(\mathbf{r}_d \| \mathbf{r}_p)$ deduce that $\mathbf{d} = \mathbf{A} \cdot \mathbf{p}$ with soundness error $\mathcal{O}\left(\epsilon_3 + \frac{N}{|\mathbb{F}|} + \epsilon_{\mathsf{eval}}(l_N)\right)$.
– Similarly, $\{\widetilde{\mathsf{vv}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_N}} \subseteq \{\widetilde{\mathsf{vx}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_n}}$ from Π_4 implies tuple lookup $\{(p_{j-1}+1, v_j)\}_{j \in [N]} \subseteq \{(j, s_j)\}_{j \in [n+1]}$ with soundness error $\mathcal{O}\left(\epsilon_4 + \frac{n \cdot N}{|\mathbb{F}|}\right)$.
– Lookup argument $\{\widetilde{\mathsf{vc}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_N}} \subseteq \{\tilde{h}_{\mathsf{diff}}(\mathbf{i})\}_{\mathbf{i} \in \{0,1\}^{l_m}}$ from Π_5 implies $\{(p_j - p_{j-1}, t_j - v_j)\}_{j \in [N]} \subseteq \{(0,i)\}_{i \in [-|\Sigma|, |\Sigma|] \setminus \{0\}} \cup \{(1,0)\}$ with soundness error $\mathcal{O}\left(\epsilon_5 + \frac{N \cdot |\Sigma|}{|\mathbb{F}|}\right)$ as $2^{l_N} = N$ and $2^{l_m} = m = \mathcal{O}(|\Sigma|)$.
– Evaluating $\tilde{f}_d(\mathbf{r}_d) = e_d$ and $\tilde{f}_p(\mathbf{1}) < n$ hold with soundness error $\mathcal{O}(\epsilon_{\mathsf{eval}}(l_N))$.

By using Theorem 2 and union bound, the extracted components with the above-satisfied properties imply $\mathbf{s} \not\preccurlyeq \mathbf{t}$ with the soundness error (total soundness error from above analysis) bounded by $\mathcal{O}\left(\frac{n \cdot N + n \cdot |\Sigma| + N \cdot |\Sigma|}{|\mathbb{F}|} + \sum_{i \in [5]} \epsilon_i\right)$. □

4.4 Efficiency

Recall notations $\mathsf{cs}, \mathsf{ps}, \mathsf{tp}, \mathsf{tv}$ and ϵ_{eval} from Section 2.4. Then, the efficiency of $\Pi_{\mathsf{non\text{-}subseq}}$ is computed based on the efficiency of component protocols $\Pi_1, \ldots, \Pi_5$ as follows.

– *Input size.* $\mathcal{O}(\mathsf{cs}(l_n) + \mathsf{cs}(l_N) + \mathsf{cs}(l_\Sigma))$.
– *Proof size.* $\mathcal{O}(\mathsf{cs}(l_N) + l_n + l_N + l_\Sigma + \mathsf{ps}(l_n) + \mathsf{ps}(l_N) + \mathsf{ps}(l_\Sigma))$ since there is one commitment to $\tilde{g}_r(\boldsymbol{X}) \in \mathbb{F}[l_N]$ sent to verifier, sumcheck protocols are bounded by $\mathcal{O}(l_n + l_N + l_\Sigma)$ rounds, and evaluations are from polynomials over l_n, l_N and l_Σ variables.
– *Prover time.* $\mathcal{O}(n + N + |\Sigma| + \mathsf{tp}(l_n) + \mathsf{tp}(l_N) + \mathsf{tp}(l_\Sigma))$ since all sums from sumcheck protocols are bounded by $\mathcal{O}(n + N + |\Sigma|)$ addends.
– *Verifier time.* $\mathcal{O}(l_n + l_N + l_\Sigma + \mathsf{tv}(l_n) + \mathsf{tv}(l_N) + \mathsf{tv}(l_\Sigma))$ since the verifier only verifies low-constant-degree univariate polynomials in the rounds of protocols from sumcheck and evaluations of polynomials over l_n, l_N, and l_Σ variables.

Instantiation. When instantiating with Sona PCS [22] (see Table 1), we achieve

– *Input size.* $\mathcal{O}(1)\mathbb{H}$.
– *Proof size.* $\mathcal{O}(l_n + l_N + l_\Sigma)\mathbb{F}$, $\mathcal{O}(1)\mathbb{H}$ and $\mathcal{O}(1)\mathbb{G}$.
– *Prover time.* $\mathcal{O}(n + N + |\Sigma|)\mathbb{F}$, $\mathcal{O}(\sqrt{n} + \sqrt{N} + \sqrt{|\Sigma|})\mathbb{G}$.
– *Verifier time.* $\mathcal{O}(\sqrt{n} + \sqrt{N} + \sqrt{|\Sigma|})\mathbb{G}$.

$\underline{\Pi_{\mathsf{non\text{-}subseq}}}$:

Inputs: The inputs to this protocol include:

- *Common Parameters:* $n, N \in \mathbb{Z}_+$, $\Sigma \subseteq \mathbb{F}$, $\Sigma_\perp = \Sigma \cup \{\perp\}$, $m = \mathcal{O}(|\Sigma|) \in \mathbb{Z}_+$, $l_n = \log_2(n+1)$, $l_N = \log_2 N$, l_Σ s.t. $2^{l_\Sigma} \geq |\Sigma_\perp|$, $l_m = \log_2 m \in \mathbb{Z}_+$.
- *Common Inputs:*
  - Common polynomials $\widetilde{\mathsf{inc}}_{l_N}(\boldsymbol{X}) \in \mathbb{F}[l_N]$, $\tilde{g}_A(\boldsymbol{Y}\|\boldsymbol{X}) \in \mathbb{F}[2l_N], \tilde{g}_{\mathsf{ch}}(\boldsymbol{X}) \in \mathbb{F}[l_\Sigma], \tilde{g}_{\mathsf{ch}'}(\boldsymbol{X}) \in \mathbb{F}[l_n], \tilde{g}_\Sigma(\boldsymbol{X}) \in \mathbb{F}[l_\Sigma], \tilde{g}_{\mathsf{iz}}(\boldsymbol{X}) \in \mathbb{F}[l_m], \tilde{g}_{\mathsf{diff}}(\boldsymbol{X}) \in \mathbb{F}[l_m]$;
  - Commitments $\sigma(\tilde{f}_s)$, $\sigma(\tilde{f}_t)$ to $\tilde{f}_s(\boldsymbol{X}), \tilde{f}_t(\boldsymbol{X})$, respectively, and $\sigma(\widetilde{\mathsf{inc}}_{l_N})$, $\sigma(\tilde{g}_A)$ (sparse), $\sigma(\tilde{g}_{\mathsf{ch}})$, $\sigma(\tilde{g}_{\mathsf{ch}'})$, $\sigma(\tilde{g}_\Sigma)$, $\sigma(\tilde{g}_{\mathsf{iz}})$, $\sigma(\tilde{g}_{\mathsf{diff}})$ to common polynomials.
- $\mathcal{P}$*'s Inputs:* $\tilde{f}_s(\boldsymbol{X}) \in \mathbb{F}[l_n]$, $\tilde{f}_t(\boldsymbol{X}) \in \mathbb{F}[l_N]$, and auxiliary inputs for evaluating PCSs.

Goal: All constraints in system (9) hold.

Pre-Processing: $\mathcal{P}$ determines $\tilde{f}_p(\boldsymbol{X}), \tilde{f}_d(\boldsymbol{X}), \tilde{f}_v(\boldsymbol{X}) \in \mathbb{F}[l_N]$, commits and sends to $\mathcal{V}$ commitments $\sigma(\tilde{f}_p)$, $\sigma(\tilde{f}_d)$ and $\sigma(\tilde{f}_v)$.

Execution: This protocol works as follows:

1. $\mathcal{V}$ samples $\beta \xleftarrow{\$} \mathbb{F}^\star$ and $\mathbf{r}_d \xleftarrow{\$} \mathbb{F}^{l_N}$ and sends $(\beta, \mathbf{r}_d)$ to $\mathcal{P}$.
2. $\mathcal{P}$ evaluates $e_d := \tilde{f}_d(\mathbf{r}_d)$ and $\tilde{g}_r(\boldsymbol{X}) = \tilde{g}_A(\mathbf{r}_d\|\boldsymbol{X})$. Commit to $\tilde{g}_r(\boldsymbol{X})$ to obtain $\sigma(\tilde{g}_r)$. Sends e_d and $\sigma(\tilde{g}_r)$ to $\mathcal{V}$.
3. Both $\mathcal{P}$ and $\mathcal{V}$ determine combined polynomials (see (11), (12), (15) and (16))
   $\tilde{h}_\Sigma(\boldsymbol{X}) = \tilde{g}_{\mathsf{ch}}(\boldsymbol{X}) + \tilde{g}_\Sigma(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_\Sigma]$, $\tilde{h}_{\mathsf{diff}}(\boldsymbol{X}) = \tilde{g}_{\mathsf{iz}}(\boldsymbol{X}) + \tilde{g}_{\mathsf{diff}}(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_m]$,
   $\widetilde{\mathsf{vs}}(\boldsymbol{X}) = \tilde{g}_{\mathsf{ch}'}(\boldsymbol{X}) + \tilde{f}_s(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_n]$, $\widetilde{\mathsf{vt}}(\boldsymbol{X}) = 1 + \tilde{f}_t(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_N]$,
   $\widetilde{\mathsf{vv}}(\boldsymbol{X}) = (\tilde{f}_d(\boldsymbol{X}) + 1) + \tilde{f}_v(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_N]$, $\widetilde{\mathsf{vx}}(\boldsymbol{X}) = \widetilde{\mathsf{inc}}_{l_n}(\boldsymbol{X}) + \tilde{f}_s(\boldsymbol{X}) \cdot \beta \in \mathbb{F}[l_n]$,
   $\widetilde{\mathsf{vc}}(\boldsymbol{X}) = (\tilde{f}_p(\boldsymbol{X}) - \tilde{f}_d(\boldsymbol{X})) + (\tilde{f}_t(\boldsymbol{X}) - \tilde{f}_v(\boldsymbol{X})) \cdot \beta \in \mathbb{F}[l_N]$.
   Here, an evaluation of the combined polynomial is computed through the evaluations of its component polynomials. For example, to evaluate $\widetilde{\mathsf{vs}}(\mathbf{r})$ for some $\mathbf{r} \in \mathbb{F}^{l_n}$, both parties run the evaluation protocol w.r.t. polynomial commitments $\sigma(\tilde{g}_{\mathsf{ch}})$ and $\sigma(\tilde{f}_s)$ to evaluate $e_{\mathsf{ch}} = \tilde{g}_{\mathsf{ch}}(\mathbf{r})$ and $e_s = \tilde{f}_s(\mathbf{r})$, and then compute $\widetilde{\mathsf{vs}}(\mathbf{r}) = e_{\mathsf{ch}} + e_s \cdot \beta$.
4. Both $\mathcal{P}$ and $\mathcal{V}$ run (independently and in parallel) sumcheck-related protocols:
   - Π_1: run Π_{lkup} for $\{\widetilde{\mathsf{vs}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_n}} \subseteq \{\tilde{h}_\Sigma(\mathbf{i})\}_{\mathbf{i} \in \{0,1\}^{l_\Sigma}}$.
   - Π_2: run Π_{lkup} for $\{\widetilde{\mathsf{vt}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_N}} \subseteq \{\tilde{h}_\Sigma(\mathbf{i})\}_{\mathbf{i} \in \{0,1\}^{l_\Sigma}}$.
   - Π_3: run Π_{sum} for $e_d = \sum_{\mathbf{j} \in \{0,1\}^{l_N}} \tilde{g}_r(\mathbf{j}) \cdot \tilde{f}_p(\mathbf{j})$.
   - Π_4: run Π_{lkup} for $\{\widetilde{\mathsf{vv}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_N}} \subseteq \{\widetilde{\mathsf{vx}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_n}}$.
   - Π_5: run Π_{lkup} for $\{\widetilde{\mathsf{vc}}(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_N}} \subseteq \{\tilde{h}_{\mathsf{diff}}(\mathbf{i})\}_{\mathbf{i} \in \{0,1\}^{l_m}}$.
   - Run evaluation protocol of PCS to check whether $e_d = \tilde{f}_d(\mathbf{r}_d)$, $\tilde{g}_r(\mathbf{r}_p) = \tilde{g}_A(\mathbf{r}_d\|\mathbf{r}_p)$, where $\mathbf{r}_p$ is random string in the end of Π_3, and $\tilde{f}_p(\mathbf{1}) < n$ from $\sigma(\tilde{f}_d)$, $\sigma(\tilde{f}_p)$, $\sigma(\tilde{g}_r)$ and sparse $\sigma(\tilde{g}_A)$.
5. $\mathcal{V}$ accepts if $\Pi_1, \ldots, \Pi_5$ are accepted and the evaluations are satisfied.

Fig. 3. Protocol $\Pi_{\mathsf{non\text{-}subseq}}$.

Acknowledgments

The work of San Ling and Huaxiong Wang was supported by Singapore Ministry of Education Academic Research Fund Tier 2 Grant T2EP20223-0028. This research is supported by the National Research Foundation, Singapore, and Infocomm Media Development Authority under its Trust Tech Funding Initiative, Strategic Capability Research Centres Funding Initiative, and Future Communications Research & Development Programme. Any opinions, findings,

and conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, and Infocomm Media Development Authority. We also thank Dr Hong Hanh Tran, Minh Pham, Dr Chan Nam Ngo, Hien Chu, and anonymous reviewers for reviewing and giving valuable comments in this result.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18**(6), 333–340 (1975). https://doi.org/10.1145/360825.360855
2. Angel, S., Ioannidis, E., Margolin, E., Setty, S., Woods, J.: Reef: Fast Succinct Non-Interactive Zero-Knowledge Regex Proofs. In: 33rd USENIX Security Symposium – USENIX Security 2024. USENIX Association (2024), https://www.usenix.org/conference/usenixsecurity24/presentation/angel
3. Bootle, J., Cerulli, A., Groth, J., Jakobsen, S., Maller, M.: Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution. In: Advances in Cryptology – ASIACRYPT 2018. Lecture Notes in Computer Science, vol. 11272, pp. 595–626. Springer International Publishing (2018). https://doi.org/10.1007/978-3-030-03326-2_20
4. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK Compilers. In: Advances in Cryptology – EUROCRYPT 2020. Lecture Notes in Computer Science, vol. 12105, pp. 677–706. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-45721-1_24
5. Campanelli, M., Faonio, A., Fiore, D., Li, T., Lipmaa, H.: Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees. In: Public-Key Cryptography – PKC 2024. Lecture Notes in Computer Science, vol. 14602, pp. 337–369. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-57722-2_11
6. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In: Advances in Cryptology – EUROCRYPT 2023. Lecture Notes in Computer Science, vol. 14005, pp. 499–530. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-30617-4_17
7. Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763 (2022), https://eprint.iacr.org/2022/1763
8. Fiat, A., Shamir, A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Advances in Cryptology — CRYPTO 1986. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer Berlin Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
9. Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315 (2020), https://eprint.iacr.org/2020/315
10. GeeksforGeeks: Subsequence meaning in dsa. online (2023), https://www.geeksforgeeks.org/subsequence-meaning-in-dsa/
11. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and Succinct NIZKs without PCPs. In: Advances in Cryptology – EUROCRYPT 2013. Lecture Notes in Computer Science, vol. 7881, pp. 626–645. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37

12. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing – STOC 2011. p. 99–108. Association for Computing Machinery (2011). https://doi.org/10.1145/1993636.1993651
13. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing – STOC 1985. p. 291–304. Association for Computing Machinery (1985). https://doi.org/10.1145/22145.22178
14. Haböck, U.: Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530 (2022), https://eprint.iacr.org/2022/1530
15. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: Advances in Cryptology - ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 177–194. Springer Berlin Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
16. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In: Advances in Cryptology – CRYPTO 2022. Lecture Notes in Computer Science, vol. 13510, pp. 359–388. Springer Nature Switzerland (2022). https://doi.org/10.1007/978-3-031-15985-5_13
17. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. In: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science – FOCS 1990. vol. 1, pp. 2–10. IEEE (1990). https://doi.org/10.1109/FSCS.1990.89518
18. Luo, N., Weng, C., Singh, J., Tan, G., Piskac, R., Raykova, M.: Privacy-preserving regular expression matching using nondeterministic finite automata. Cryptology ePrint Archive, Paper 2023/643 (2023), https://eprint.iacr.org/2023/643
19. Nguyen, W., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A Scalable Framework for Folding-based SNARKs. In: Advances in Cryptology – CRYPTO 2024 (2024), to appear
20. Raymond, M., Evers, G., Ponti, J., Krishnan, D., Fu, X.: Efficient Zero Knowledge for Regular Language. In: 19th EAI International Conference on Security and Privacy in Communication Networks – SecureComm 2023 (2023), to appear
21. Setty, S.: Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In: Advances in Cryptology – CRYPTO 2020. Lecture Notes in Computer Science, vol. 12172, pp. 704–737. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-56877-1_25
22. Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with lasso. In: Advances in Cryptology – EUROCRYPT 2024. Lecture Notes in Computer Science, vol. 14656, pp. 180–209. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-58751-1_7
23. Thakur, S.: A flexible snark via the monomial basis. Cryptology ePrint Archive, Paper 2023/1255 (2023), https://eprint.iacr.org/2023/1255
24. Thompson, K.: Programming techniques: Regular expression search algorithm. Commun. ACM **11**(6), 419–422 (1968). https://doi.org/10.1145/363347.363387
25. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-Efficient zkSNARKs Without Trusted Setup. In: 2018 IEEE Symposium on Security and Privacy – S&P 2018. pp. 926–943. IEEE (2018). https://doi.org/10.1109/SP.2018.00060
26. Zhang, C., DeStefano, Z., Arun, A., Bonneau, J., Grubbs, P., Walfish, M.: Zombie: Middleboxes that Don't Snoop. In: 21st USENIX Symposium on Networked

Systems Design and Implementation – NSDI 2024. pp. 1917–1936. USENIX Association (2024), https://www.usenix.org/conference/nsdi24/presentation/zhang-collin
27. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In: 2020 IEEE Symposium on Security and Privacy – S&P 2020. pp. 859–876. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00052
28. Zhang, Y., Sun, S.F., Gu, D.: Efficient KZG-Based Univariate Sum-Check and Lookup Argument. In: Public-Key Cryptography – PKC 2024. Lecture Notes in Computer Science, vol. 14602, pp. 400–425. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-57722-2_13
29. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In: 2017 IEEE Symposium on Security and Privacy – S&P 2017. pp. 863–880. IEEE (2017). https://doi.org/10.1109/SP.2017.43

A Proof of Lemma 1

Proof (Proof of Lemma 1). We prove this lemma by induction.

For the base case with $p_0 = 0$, it is trivial since an empty string is a subsequence of an empty string.

Assume by inductive hypothesis that, for $j \in [N-1]$, p_{j-1} is the maximum index satisfying

$$(s_1, \ldots, s_{p_{j-1}}) \lhd (t_1, \ldots, t_{j-1}).$$

Now, we prove that this is also true w.r.t. p_j, i.e., the maximum index satisfying $(s_1, \ldots, s_{p_j}) \lhd (t_1, \ldots, t_j)$. We claim that $p_{j-1} \leq p_j \leq p_{j-1} + 1$. It is trivial to see that $p_{j-1} \leq p_j$ since a subsequence to $(t_1, \ldots, t_{j-1})$ is also a subsequence of $(t_1, \ldots, t_j)$. What happens if $p_j \geq p_{j-1} + 2$?

Since $(s_1, \ldots, s_{p_{j-1}}) \lhd (t_1, \ldots, t_{j-1})$, we know that there exist $\mathsf{id}_1, \ldots, \mathsf{id}_{p_{j-1}}$ satisfying

$$\begin{cases} \mathsf{id}_1 < \cdots < \mathsf{id}_{p_{j-1}} \leq j-1, \\ (s_1, \ldots, s_{p_{j-1}}) = (t_{\mathsf{id}_1}, \ldots, t_{\mathsf{id}_{p_{j-1}}}). \end{cases}$$

Hence, if $p_j \geq p_{j-1}+2$, we know that there exist $\mathsf{id}_{p_{j-1}+1}$ and $\mathsf{id}_{p_{j-1}+2}$ satisfying

$$\begin{cases} \mathsf{id}_1 < \cdots < \mathsf{id}_{p_{j-1}+2} \leq j, \\ (s_1, \ldots, s_{p_{j-1}+2}) = (t_{\mathsf{id}_1}, \ldots, t_{\mathsf{id}_{p_{j-1}+2}}). \end{cases}$$

If $\mathsf{id}_{p_{j-1}+1} \leq j-1$, then it contradicts to the fact that p_{j-1} is the maximum index satisfying $(s_1, \ldots, s_{p_{j-1}}) \lhd (t_1, \ldots, t_{j-1})$. Hence, $j - 1 < \mathsf{id}_{p_{j-1}+1} \leq j$. Therefore, we also have $j - 1 < \mathsf{id}_{p_{j-1}+1} < \mathsf{id}_{p_{j-1}+2} \leq j$, a contradiction since there cannot exist two distinct integers in $(j-1, j]$. Thus, $p_{j-1} \leq p_j \leq p_{j-1}+1$.

With the above argument, we know that, if $p_j = p_{j-1} + 1$, then it must hold that $\mathsf{id}_{p_{j-1}+1} = \mathsf{id}_{p_j} = j$ which only happens when $s_{p_{j-1}+1} = t_j$. Thus, we deduce that $p_j = p_{j-1} + 1$, if $s_{p_{j-1}+1} = t_j$, and $p_j = p_{j-1}$, otherwise. We hence conclude the proof. □

B Additional Preliminaries

B.1 Security of SNARKs

In this appendix, we recall the security of SNARKs. A SNARK for an NP language $\mathcal{L}$ (with respect to some NP relation $\mathcal{R}$) must satisfy the following.

- *Completeness.* Given $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, for any statement $x \in \mathcal{L}$, there exists a witness w s.t.

$$\Pr\left[\mathcal{V}(\mathsf{pp}, x, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathcal{R}(x, w) = 1, \\ \pi \leftarrow \mathcal{P}(\mathsf{pp}, x, w) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

  where $\mathsf{negl}(\lambda)$ is a negligible function of λ.
- *Knowledge Soundness.* For any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$ s.t. for any x, it holds that

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{pp}, x, \pi^*) = 1 \\ \wedge \mathcal{R}(x, w') = 0 \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \pi^\star \leftarrow \mathcal{A}(\mathsf{pp}, x), \\ w' \leftarrow \mathcal{E}(\mathsf{pp}, x, \pi^\star) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

  where $\mathsf{negl}(\lambda)$ is a negligible function of λ.
- *Succinctness.* The communication between $\mathcal{P}$ and $\mathcal{V}$ is sub-linear in the size of $x \in \mathcal{L}$.
- *Sublinear Verification Time.* The verification time is sublinear in $|x|$.

B.2 Security of Polynomial Commitment Schemes

In this appendix, we describe the security of PCSs. An extractable PCS

$$\mathsf{PCS} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$$

is secure if it satisfies completeness, binding, and knowledge soundness as follows.

- *Completeness.* For any l-variate multilinear polynomial $f(\boldsymbol{X}) \in \mathbb{F}[l]$, the following probability is overwhelming by in λ, i.e., at least $1 - \mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{l} \mathsf{Eval}\,\langle \mathcal{P}(f(\boldsymbol{X}), \mathsf{aux}), \mathcal{V}(\mathbf{r})\rangle\,(\mathsf{pp}, \sigma(f), S) = 1 \\ \wedge S = f(\mathbf{r}) \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\sigma(f), \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{pp}, f(\boldsymbol{X})) \end{array}\right].$$

- *Binding.* For any PPT adversary $\mathcal{A}$, the following probability is negligible in λ, i.e., at most $\mathsf{negl}(\lambda)$:

$$\Pr\left[b_0 = b_1 = 1 \wedge f_0(\boldsymbol{X}) \neq f_1(\boldsymbol{X}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\sigma, f_0(\boldsymbol{X}), f_1(\boldsymbol{X}), \mathsf{aux}_0, \mathsf{aux}_1) \leftarrow \mathcal{A}(\mathsf{pp}), \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \sigma, f_0(\boldsymbol{X}), \mathsf{aux}_0), \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \sigma, f_1(\boldsymbol{X}), \mathsf{aux}_1) \end{array}\right].$$

- *Knowledge Soundness.* Given $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$, Eval is a succinct argument of knowledge for NP relation

$$\mathcal{R}_{\mathsf{eval}} = \left\{ \big(\sigma(f), \mathbf{r}, e;\ f(\boldsymbol{X}), \mathsf{aux}\big) \colon \begin{array}{l} f(\boldsymbol{X}) \in \mathbb{F}[l] \wedge f(\mathbf{r}) = e \\ \wedge \mathsf{Open}(\mathsf{pp}, \sigma(f), f(\boldsymbol{X}), \mathsf{aux}) = 1 \end{array} \right\}.$$

$\underline{\Pi_{\mathsf{sum}}}$:

Parties: Denote by $\mathcal{P}$ and $\mathcal{V}$ the prover and verifier, respectively.
Inputs: The inputs to this protocol include:
- *Common Parameters:* Integer $l \in \mathbb{Z}_+$.
- *Common Inputs:* polynomial commitment $\sigma(f)$ and sum $S \in \mathbb{F}$.
- $\mathcal{P}$*'s Input:* $f(\boldsymbol{X}) \in \mathbb{F}[l]$ and auxiliary input supporting evaluating from $\sigma(f)$.

Goal: $\sum_{\mathbf{i}\in\{0,1\}^l} f(\mathbf{i}) = S$.
Execution: This protocol runs as follows:

1. $\mathcal{P}$ sends polynomial
$$f_1(X_1) = \sum_{\mathbf{i}\in\{0,1\}^{l-1}} f(X_1\|\mathbf{i}).$$
$\mathcal{V}$ rejects if $f_1(0) + f_1(1) \neq S$ and degree of $f_1(X_1)$, denoted by $\deg_1$, equals to that of X_1 in $f(\boldsymbol{X})$, denoted by $\deg(X_1)$. $\mathcal{V}$ chooses sample $r_1 \xleftarrow{\$} \mathbb{F}$ and sends r_1 to $\mathcal{P}$.
2. For each j runs sequentially from 2 to l, $\mathcal{P}$ sends the polynomial
$$f_j(X_j) = \sum_{\mathbf{i}\in\{0,1\}^{l-j}} f((r_1,\ldots,r_{j-1},X_j)\|\mathbf{i}).$$
$\mathcal{V}$ rejects if $f_j(0) + f_j(1) \neq f_{j-1}(r_{j-1})$ and degree $\deg_j = \deg(X_j)$. $\mathcal{V}$ samples $r_j \xleftarrow{\$} \mathbb{F}$ and sends r_j to $\mathcal{P}$.
3. $\mathcal{V}$ accepts if $e = f(r_1,\ldots,r_l)$, where $e = f_l(r_l)$, by running protocol Eval of the respective PCS. This protocol also returns $(\mathbf{r}, e)$ where $\mathbf{r} = (r_1,\ldots,r_l)$.

Fig. 4. Protocol Π_{sum}.

B.3 Protocols from Sumcheck

Sumcheck Protocol. The sumcheck protocol Π_{sum} for relation

$$\mathcal{R}_{\mathsf{sum}} = \left\{ \left(S, \sigma(f);\ f(\boldsymbol{X}) \in \mathbb{F}[l], \mathsf{aux} \right) :\ S = \textstyle\sum_{\mathbf{i}\in\{0,1\}^l} f(\mathbf{i}) \right\},$$

recalled from (5), proceeds as in Figure 4. We use the following Lemma 3 to show its security.

Lemma 3 ([17]). *For an l-variate polynomial $f(\boldsymbol{X}) \in \mathbb{F}[l]$ of degree ρ, the above sumcheck protocol is perfectly complete and sound with soundness error $\mathcal{O}\left((l\cdot\rho)/|\mathbb{F}| + \epsilon_{\mathsf{eval}}(l)\right)$ where $\epsilon_{\mathsf{eval}}(l)$ is introduced in Section 2.4.*

Table 4. This table contains communication cost, soundness error, prover time and verifier time of sumcheck protocol Π_{sum} (c.f. [6]) for polynomial $f(\boldsymbol{X}) = h(g_1(\boldsymbol{X}),\ldots,g_c(\boldsymbol{X})) \in \mathbb{F}[l]$ where $c \in \mathbb{Z}_+$ is a constant, $g_c(\boldsymbol{X})$ is of degree 1 and $h(\cdot)$ is of degree ρ. Notations $\mathsf{cs}, \mathsf{ps}, \mathsf{tp}, \mathsf{tv}$ and ϵ_{eval} are introduced in Section 2.4.

Proof size	**Soundness error**	**Prover time**	**Verifier time**
$\mathcal{O}(l\cdot\rho + \mathsf{ps}(l))$	$\mathcal{O}\left(\frac{l\cdot\rho}{\lvert\mathbb{F}\rvert} + \epsilon_{\mathsf{eval}}(l)\right)$	$\mathcal{O}(2^l\rho\log_2^2\rho + \mathsf{tp}(l))$	$\mathcal{O}(l\cdot\rho + \mathsf{tv}(l))$

In Table 4, we recall the communication cost, soundness error, prover and verifier time of sumcheck protocol Π_{sum} from [6], w.r.t.

$$f(\boldsymbol{X}) = h(g_1(\boldsymbol{X}), \ldots, g_c(\boldsymbol{X})) \in \mathbb{F}[l]$$

where $c \in \mathbb{Z}_+$ is a constant, $g_c(\boldsymbol{X})$ is of degree 1 and $h(\cdot)$ is of degree ρ.

Lookup Argument Adapted from [14]. As said above, as we are only interested in the lookup protocol from [14], in this appendix, we only consider the lookup protocol adapted from [14]. Other lookup protocols can be found in [7,22]. Let $l_a, l_b \in \mathbb{Z}_+$. Protocol Π_{lkup} in Figure 5 aims to show the satisfaction of relation

$$\mathcal{R}_{\mathsf{lkup}} = \left\{ \big(\sigma(\tilde{f}_a), \sigma(\tilde{f}_b);\ \tilde{f}_a(\boldsymbol{X}), \tilde{f}_b(\boldsymbol{X}), \mathsf{aux} \big) \colon \{\tilde{f}_a(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_a}} \subseteq \{\tilde{f}_b(\mathbf{i})\}_{\mathbf{i} \in \{0,1\}^{l_b}} \right\},$$

recalled from (6), where $\tilde{f}_a(\boldsymbol{X}) \in \mathbb{F}[l_a]$ and $\tilde{f}_b(\boldsymbol{X}) \in \mathbb{F}[l_b]$ are multilinear. According to [14], $\{\tilde{f}_a(\mathbf{j})\}_{\mathbf{j} \in \{0,1\}^{l_a}} \subseteq \{\tilde{f}_b(\mathbf{i})\}_{\mathbf{i} \in \{0,1\}^{l_b}}$ iff there exists $\widetilde{\mathsf{mul}}(\boldsymbol{X}) \in \mathbb{F}[l_b]$ satisfying

$$\sum_{\mathbf{j} \in \{0,1\}^{l_a}} \frac{1}{\tilde{f}_a(\mathbf{j}) + Y} = \sum_{\mathbf{i} \in \{0,1\}^{l_b}} \frac{\widetilde{\mathsf{mul}}(\mathbf{i})}{\tilde{f}_b(\mathbf{i}) + Y}. \tag{17}$$

The intuition for Π_{lkup} for relation $\mathcal{R}_{\mathsf{lkup}}$ is as follows. To show (17), prover instead shows the reduced formula

$$\sum_{\mathbf{j} \in \{0,1\}^{l_a}} \frac{1}{\tilde{f}_a(\mathbf{j}) + \gamma} = T \text{ and } \sum_{\mathbf{i} \in \{0,1\}^{l_b}} \frac{\widetilde{\mathsf{mul}}(\mathbf{i})}{\tilde{f}_b(\mathbf{i}) + \gamma} = T, \tag{18}$$

where $\gamma \xleftarrow{\$} \mathbb{F}$ given by verifier, holds with completeness error $\mathcal{O}\left(\frac{2^{l_a}+2^{l_b}}{|\mathbb{F}|}\right)$ (due to potential division by zero) and soundness error $\mathcal{O}\left(\frac{2^{l_a+l_b}}{|\mathbb{F}|}\right)$ (i.e., the original formula is false but reduced formula holds). Then, prover computes $\tilde{f}'_a(\boldsymbol{X})$ and $\tilde{f}'_b(\boldsymbol{X})$ encoding $\left((\tilde{f}_a(\mathbf{j}) + \gamma)^{-1}\right)_{\mathbf{j} \in \{0,1\}^{l_a}}$ and $\left(\widetilde{\mathsf{mul}}(\mathbf{i}) \cdot (\tilde{f}_b(\mathbf{i}) + \gamma)^{-1}\right)_{\mathbf{j} \in \{0,1\}^{l_b}}$, respectively. Then, prover must show that $\tilde{f}'_a(\boldsymbol{X})$ and $\tilde{f}'_b(\boldsymbol{X})$ are computed honestly. Hence, prover show that $\tilde{f}'_a(\mathbf{j}) \cdot (\tilde{f}_a(\mathbf{j}) + \gamma)^{-1} - 1 = 0$ for all $\mathbf{j} \in \{0,1\}^{l_a}$ and $\tilde{f}'_b(\mathbf{i}) \cdot \widetilde{\mathsf{mul}}(\mathbf{i}) \cdot (\tilde{f}_b(\mathbf{i}) + \gamma)^{-1} - 1 = 0$ for all $\mathbf{i} \in \{0,1\}^{l_b}$. Hence, we apply the well-known technique to show that

$$\sum_{\mathbf{j} \in \{0,1\}^{l_a}} \left(\tilde{f}'_a(\mathbf{j}) \cdot (\tilde{f}_a(\mathbf{j}) + \gamma)^{-1} - 1\right) \cdot \widetilde{\mathsf{eq}}_{l_a}(\mathbf{r}_1 \| \mathbf{j}) = 0 \text{ and}$$
$$\sum_{\mathbf{i} \in \{0,1\}^{l_b}} \left(\tilde{f}'_b(\mathbf{i}) \cdot \widetilde{\mathsf{mul}}(\mathbf{i}) \cdot (\tilde{f}_b(\mathbf{i}) + \gamma)^{-1} - 1\right) \cdot \widetilde{\mathsf{eq}}_{l_b}(\mathbf{r}_2 \| \mathbf{i}) = 0,$$

for some $\mathbf{r}_1 \xleftarrow{\$} \mathbb{F}^{l_a}$ and $\mathbf{r}_2 \xleftarrow{\$} \mathbb{F}^{l_b}$, by running two instances of Π_{sum} for the two formulas. The use of $\widetilde{\mathsf{eq}}_{l_a}(\mathbf{r}_1 \| \boldsymbol{X})$ and $\widetilde{\mathsf{eq}}_{l_b}(\mathbf{r}_2 \| \boldsymbol{X})$ is employed in Spartan [21]

$\underline{\Pi_{\mathsf{lkup}}}$:

Parties: Denote by $\mathcal{P}$ and $\mathcal{V}$ the prover and verifier, respectively.

Inputs: The inputs to this protocol include:

- *Common Parameters:* $l_a, l_b \in \mathbb{Z}_+$.
- *Common Inputs:* Polynomial commitments $\sigma(\tilde{f}_a), \sigma(\tilde{f}_b)$.
- *$\mathcal{P}$'s Inputs:* $\tilde{f}_a(\boldsymbol{X}) \in \mathbb{F}[l_a], \tilde{f}_b(\boldsymbol{X}) \in \mathbb{F}[l_b]$ and auxiliary inputs supporting evaluating polynomial commitments in common inputs. Notice here that $\tilde{f}_a(\boldsymbol{X})$ and $\tilde{f}_b(\boldsymbol{X})$ encode vectors $\mathbf{a} = (a_j)_{j\in[0,2^{l_a}-1]} \in \mathbb{F}^{2^{l_a}}$ and $\mathbf{b} = (b_i)_{i\in[0,2^{l_b}-1]} \in \mathbb{F}^{2^{l_b}}$

Goal: $\{a_j\}_{j\in[0,2^{l_a}-1]} \subseteq \{b_i\}_{i\in[0,2^{l_b}-1]}$ by showing the existence of $\widetilde{\mathsf{mul}}(\boldsymbol{X}) \in \mathbb{F}[l_b]$ satisfying $\sum_{\mathbf{j}\in\{0,1\}^{l_a}} (\tilde{f}_a(\mathbf{j}) + Y)^{-1} = \sum_{\mathbf{i}\in\{0,1\}^{l_b}} \widetilde{\mathsf{mul}}(\mathbf{i}) \cdot (\tilde{f}_b(\mathbf{i}) + Y)^{-1}$.

Pre-Processing: $\mathcal{P}$ determines $\widetilde{\mathsf{mul}}(\boldsymbol{X}) \in \mathbb{F}[l_b]$, commits and sends to $\mathcal{V}$ polynomial commitment $\sigma(\widetilde{\mathsf{mul}})$.

Execution: This protocol runs as follows:

1. $\mathcal{V}$ samples $\gamma \xleftarrow{\$} \mathbb{F}$ and sends γ to $\mathcal{P}$.
2. $\mathcal{P}$ does the followings:
   - Compute $\tilde{f}'_v(\boldsymbol{X}) \in \mathbb{F}[l_v]$, for $v \in \{a, b\}$, to be MLE of function $f'_v : \{0,1\}^{l_v} \to \mathbb{F}$ mapping $\mathbf{i} \mapsto (f_v(\mathbf{i}) + \gamma)^{-1} \in \mathbb{F} \quad \forall \mathbf{i} \in \{0,1\}^{l_v}$.
   - Compute $T = \sum_{\mathbf{j}\in\{0,1\}^{l_a}} \tilde{f}'_a(\mathbf{j})$.
   - Compute and send polynomial commitments $\sigma(\tilde{f}'_a)$ and $\sigma(\tilde{f}'_b)$ to $\mathcal{V}$.
3. $\mathcal{V}$ samples $\mathbf{r}_1 \xleftarrow{\$} \mathbb{F}^{l_a}, \mathbf{r}_2 \xleftarrow{\$} \mathbb{F}^{l_b}$ and sends $(\mathbf{r}_a, \mathbf{r}_b)$ to $\mathcal{P}$.
4. $\mathcal{P}$ and $\mathcal{V}$ run Π_{sum} for checking whether

$$
\begin{aligned}
0 = &\sum_{\mathbf{i}\in\{0,1\}^{l_a}} \left(\tilde{f}'_a(\mathbf{i}) \cdot (\tilde{f}_a(\mathbf{i}) + \gamma) - 1\right) \cdot \widetilde{\mathsf{eq}}_{l_a}(\mathbf{r}_1 \| \mathbf{i}) \\
0 = &\sum_{\mathbf{i}\in\{0,1\}^{l_b}} \left(\tilde{f}'_b(\mathbf{i}) \cdot (\tilde{f}_b(\mathbf{i}) + \gamma) - 1\right) \cdot \widetilde{\mathsf{eq}}_{l_b}(\mathbf{r}_2 \| \mathbf{i}) \\
T = &\sum_{\mathbf{j}\in\{0,1\}^{l_a}} \tilde{f}'_a(\mathbf{j}) \text{ where } \tilde{f}'_a(\boldsymbol{X}) \text{ is of degree 1,} \\
T = &\sum_{\mathbf{i}\in\{0,1\}^{l_b}} \widetilde{\mathsf{mul}}(\mathbf{i}) \cdot \tilde{f}'_b(\mathbf{i})
\end{aligned}
$$

   For the first argument, we remind that Π_{sum} reduces the check to a pair $(\mathbf{r}_a, e_a)$ and then both parties run the evaluation protocol w.r.t. $\sigma(\tilde{f}'_a)$ and $\sigma(\tilde{f}_a)$ to check whether $e_a = \left(\tilde{f}'_a(\mathbf{r}_a) \cdot (\tilde{f}_a(\mathbf{r}_a) + \gamma) - 1\right) \cdot \widetilde{\mathsf{eq}}_{l_a}(\mathbf{r}_1 \| \mathbf{r}_a)$. Similar ways of checking apply to the remaining three executions of Π_{sum}.
5. $\mathcal{V}$ accepts if all four executions of Π_{sum} are accepted.

Fig. 5. Protocol Π_{lkup}.

and HyperPlonk [6]. This implies well-formednesses of $\tilde{f}'_a(\boldsymbol{X})$ and $\tilde{f}'_b(\boldsymbol{X})$ with high probability. Eventually, run the other two instances of Π_{sum} to show the two formulas

$$
T = \sum_{\mathbf{j}\in\{0,1\}^{l_a}} \tilde{f}'_a(\mathbf{j}) \text{ and } T = \sum_{\mathbf{i}\in\{0,1\}^{l_b}} \widetilde{\mathsf{mul}}(\mathbf{i}) \cdot \tilde{f}'_b(\mathbf{i}).
$$

Lemma 4 (Security of Π_{lkup}). *Let ϵ_{eval} be the total soundness error from invocations (in a constant number of times) to the evaluation protocol of the employed PCS incurred in Π_{lkup}. Then, protocol Π_{lkup} for relation $\mathcal{R}_{\mathsf{lkup}}$ satisfies completeness, with completeness error $\mathcal{O}\left(\frac{2^{l_a+l_b}}{|\mathbb{F}|}\right)$, and knowledge soundness, with soundness error $\mathcal{O}\left(\frac{l_a+l_b}{|\mathbb{F}|}+\epsilon_{\mathsf{eval}}(l)\right)$, where ϵ_{eval} is introduced in Section 2.4, if (i) Π_{sum} satisfies completeness and knowledge soundness, and (ii) the employed PCS is secure and extractable.*

In our result, we are only interested in polynomials $\tilde{f}_a(\boldsymbol{X})$ and $\tilde{f}_b(\boldsymbol{X})$ of degree 1. Hence, the protocol Π_{lkup} only deals with polynomials of degree at most 3. Eventually, the cost of Π_{lkup} is summarized in Table 5.

Table 5. This table contains communication cost, soundness error, prover time, and verifier time of sumcheck protocol Π_{lkup} (adapted from efficiency of Π_{sum} discussed in [6]). Notations $\mathsf{cs}, \mathsf{ps}, \mathsf{tp}, \mathsf{tv}$ and ϵ_{eval} are introduced in Section 2.4.

Proof size	**Soundness error**	**Prover time**	**Verifier time**
$\mathcal{O}(l_a + l_b$ $+\mathsf{ps}(l_a) + \mathsf{ps}(l_b))$	$\mathcal{O}\left(\frac{l_a+l_b}{\|\mathbb{F}\|} + \epsilon_{\mathsf{eval}}(l_a) + \epsilon_{\mathsf{eval}}(l_b)\right)$	$\mathcal{O}(2^{l_a} + 2^{l_b}$ $+\mathsf{tp}(l_a) + \mathsf{tp}(l_b))$	$\mathcal{O}(l_a + l_b$ $+\mathsf{tv}(l_a) + \mathsf{tv}(l_b))$