Intersection Non-Emptiness for Tree Shaped Finite Automata

Michael Wehar

University at Buffalo
Buffalo, NY, USA
mwehar@buffalo.edu

Abstract

We apply a reduction technique paralleling Chen, Huang, Kanj, and Xia (2006) to show that if intersection non-emptiness for k tree shaped automata is solvable in $n^{o(k)}$ time, then the exponential time hypothesis (ETH) is false. Then, we apply a reduction technique paralleling Williams (2005) to show that if intersection non-emptiness for two tree shaped automata is solvable in $O(n^{2-\varepsilon})$ time, then the strong exponential time hypothesis (SETH) is false. Further, we introduce a parameterized equivalence between intersection non-emptiness, weighted CNF-SAT, and the clique problem for hypergraphs.

1998 ACM Subject Classification F.1.1 Models of Computation, F1.3 Complexity Measures and Classes, F.4.3 Formal Languages

Keywords and phrases Automata, Regular Languages, Reachability, Constraint Satisfaction, Fixed Parameter Complexity

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

1.1 Background

Consider a boolean formula in conjunctive normal form. We can construct one automaton for each clause so that a given assignment satisfies all of the clauses if and only if it satisfies all of the automata. In other words, automata are used to collectively verify that a given assignment satisfies each of the clauses. In general, automata are effective devices for the collective verification of constraints. This insight leads us to intersection non-emptiness.

Given a finite list of DFA's (deterministic finite automata), does there exist a string that simultaneously satisfies all of the DFA's? This problem is known as the intersection non-emptiness problem for DFA's because it is equivalent to determining if the corresponding regular languages have a non-empty intersection. Let's abbreviate this intersection problem by $\text{IE}_{\mathcal{D}}$. Further, let n denote the total input size and k denote the number of DFA's in the list. When k is fixed, we denote the intersection non-emptiness problem by $k\text{-IE}_{\mathcal{D}}$.

Although intersection non-emptiness is motivated as a constraint satisfaction problem, it can also be viewed as a graph reachability problem. In particular, the standard solution for $\text{IE}_{\mathcal{D}}$ involves searching through a product automaton's state diagram.[1]

▶ **Proposition 1.** *For each $k \in \mathbb{N}$, $k\text{-IE}_{\mathcal{D}} \in \text{DTIME}(n^{O(k)})$.*

[1] The Cartesian product construction is a classic construction that is used for showing that regular languages are closed under intersection [18].

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sketch of proof. Let a finite list of DFA's $\{D_i\}_{i\in[k]}$ be given. We can build a single product DFA $\mathcal{D}$ such that $L(\mathcal{D}) = \bigcap_{i\in[k]} L(D_i)$. Hence, we reduced k-IE$_{\mathcal{D}}$ to determining whether $L(\mathcal{D})$ is non-empty. That is, whether there is a path from the start state to a final state in $\mathcal{D}$'s state diagram. Then, we proceed by performing a basic graph search such as breadth first search. Although breadth first search is efficient[2], $\mathcal{D}$'s state diagram may be large. If each DFA in $\{D_i\}_{i\in[k]}$ has at most n states, then in the worst case the product DFA $\mathcal{D}$ has at most n^k states. ◀

The preceding solution leads to a deterministic algorithm that runs in roughly n^k time using n^k space. Further, using a non-deterministic approach for graph reachability, we get a non-deterministic algorithm that runs in n^k time using $O(k\log(n))$ space. However, it's not known if there are any faster deterministic or non-deterministic algorithms for IE$_{\mathcal{D}}$.

There has been a whole history of hardness results for intersection non-emptiness. Kozen (1977) showed that IE$_{\mathcal{D}}$ is PSPACE-complete [11]. Next, Kasai and Iwata (1985) showed that for fixed k, solving k-IE$_{\mathcal{D}}$ requires $\Theta(k\log(n))$ non-deterministic space for Turing machines with a binary tape alphabet [10]. Then, Lange and Rossmanith (1992) showed that if we constrain k to be $\log(n)$, then IE$_{\mathcal{D}}$ becomes NSPACE($\log^2(n)$)-complete [13]. Finally, a combination of results from Karakostas, Lipton, and Viglas (2003) and Wehar (2014) showed that if we can solve k-IE$_{\mathcal{D}}$ in DTIME($n^{o(k)}$), then P $\neq$ NL [9, 23]. That is, if for every $\varepsilon > 0$ there exists k sufficiently large such that k-IE$_{\mathcal{D}} \in$ DTIME($n^{\varepsilon k}$), then P $\neq$ NL.

Hardness results have also been shown for a few notable restricted classes of DFA's. Consider the class of DFA's whose state diagrams are acyclic[3] (excluding the dead state). The intersection problem for this class of DFA's was shown to be NP-complete [19]. Further, for fixed k, this intersection problem was observed to characterize NTISP($n, k\log(n)$) [23]. Also, consider the class of DFA's that have a commutative transition monoid. The intersection problem for this class of DFA's was shown to be NP-complete even when the number of final states is fixed [4].

1.2 Our Contribution

Tree shaped automata make up a restricted class of finite automata. These automata are of particular interest to us because their elegant structure provides simplistic approaches to collectively verify constraints.

In this paper, we investigate the intersection non-emptiness problem for tree shaped automata. In Section 3, paralleling [6] and [24], we apply fine-grained reduction techniques to reduce boolean satisfiability to intersection non-emptiness. In particular, we show that (1) if intersection non-emptiness for tree shaped automata is solvable in $n^{o(k)}$ time, then the exponential time hypothesis is false and (2) if intersection non-emptiness for two tree shaped automata is solvable in $O(n^{2-\varepsilon})$ time, then the strong exponential time hypothesis is false. Finally, in Section 4, we present parameterized reductions to show that intersection non-emptiness for k tree shaped automata over an input alphabet $[c]$ is both equivalent to k-weighted satisfiability for c-CNF formulas and k-clique for c-uniform hypergraphs.

---

[2] A multi-tape Turing machine can carry out the breadth first search in roughly n^{2k} time. A random access machine can carry out the search in roughly n^k time.

[3] An acyclic DFA is essentially an oblivious binary decision diagram (as in [1]).

2 Preliminaries

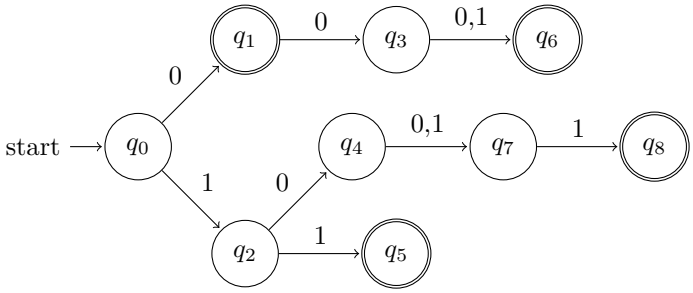
2.1 Tree Shaped Automata

In this section, we introduce tree shaped automata and their basic structural properties.

Finite automata are typically expressed by directed labeled graphs called state diagrams. When drawn, it is common to compress state diagrams by allowing multi-label transitions. A multi-label transition consists of a source state, a target state, and a finite set of alphabet characters. If any one of the characters in the finite set is read, then the automaton can move from the source state to the target state. When considering state diagrams, we will assume that all transitions with the same source and target states are compressed to a single multi-label transition.

An automaton is said to be **tree shaped**[4] if its state diagram (without the dead state) forms a rooted tree such that (1) the root of the tree is the start state, (2) all transitions are directed towards the leaves, and (3) there are no loops.

Consider the following example of a tree shaped DFA D_1:



Since tree shaped DFA's don't contain any loops or directed cycles, they can only accept finite languages. Notice that D_1 accepts the language $\{0, 11, 000, 001, 1001, 1011\}$.

A **branch** is a path from the root to a leaf. For example, the branches of D_1 are $\{0, 11, 00*, 10*1\}$ where an asterisk abbreviates a multi-label 0 or 1 transition. Since the accepting paths are exactly the branches, we can simply represent any tree shaped automata by the corresponding set of branches.

2.2 Pruned Product Construction

The Cartesian product construction is a classic construction that is used for showing that regular languages are closed under intersection [18]. The idea is that given DFA's D_1 and D_2, there exists a product DFA $\mathcal{D}$ such that $L(\mathcal{D}) = D_1 \cap D_2$. For tree shaped DFA's, we consider a **pruned product** construction. Let tree shaped DFA's D_1 and D_2 be given. First, apply the product construction to D_1 and D_2. Next, prune away the dead branches and unreachable states to get a tree shaped DFA $\mathcal{D}$ such that $L(\mathcal{D}) = D_1 \cap D_2$.

▶ **Lemma 2.** *Let tree shaped DFA's D_1 and D_2 be given. If D_1 has m_1 states of height h and D_2 has m_2 states of height h, then the pruned product of D_1 and D_2 is a tree shaped DFA with at most $m_1 \cdot m_2$ states of height h.*

[4] A tree shaped DFA is essentially a reduced form of an oblivious binary decision tree. Related concepts have been explored in [16].

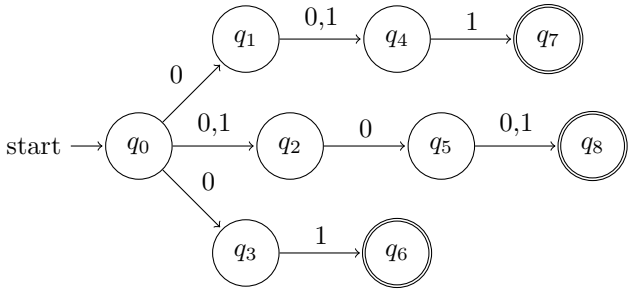
Sketch of proof. Consider the product of D_1 and D_2. The reachable states from the product automaton form a tree. One can show this by arguing that the indegree of any product state (s_1, s_2) is at most one because the indegree of s_1 in D_1 is at most one and the indegree of s_2 in D_2 is at most one. Further, one can prune away any branches that don't lead to a final state to get a tree shaped DFA. This is the pruned product of D_1 and D_2.

The pruned product is leveled in the sense that each state has a height relative to the start state. For a given state (s_1, s_2) of height h, s_1 must be of height h in D_1 and s_2 must be of height h in D_2. Since there are at most m_1 states of height h in D_1 and at most m_2 states of height h in D_2, there are at most $m_1 \cdot m_2$ states of height h in the pruned product. ◀

2.3 Non-Deterministic Automata

We've seen that any tree shaped automaton can be represented by the corresponding set of branches. Similarly, every set of branches has a corresponding tree shaped automaton, but this automaton might not be deterministic.

For example, consider the set of branches $\{0*1, *0*, 01\}$. There is no tree shaped DFA with these three branches. However, the following is the corresponding tree shaped NFA that trivially branches out from the start state:



In general, there is an exponential blow-up when converting from an NFA to an equivalent DFA. Similarly, there is an exponential blow-up when converting from a tree shaped NFA to an equivalent tree shaped DFA.

▶ **Proposition 3.** *There exists a class of languages $\{L_n\}_{n\in\mathbb{N}}$ such that for each $n \in \mathbb{N}$, L_n is accepted by some tree shaped NFA with at most $2n^2 + 1$ states. However, the smallest DFA that accepts L_n has at least 2^n states.*

Sketch of proof. For each $n \in \mathbb{N}$, consider the following language:

$$L_n = \{\, x \cdot y \mid x, y \in \{0,1\}^n \text{ and } \exists\, i \in [n] \text{ such that } x_i = y_i \}.$$

In other words, L_n is the set of strings of length $2n$ where the bit at position i is the same as the bit at position $n + i$ for some $i \in [n]$. We can build a tree shaped NFA for L_n where each branch represents one of n possible values for i. This NFA will have n branches of length $2n$ each. As a result, the NFA will have size at most $2n^2 + 1$.

For each string x of length n, consider the set S_x such that $y \in S_x$ if and only if $x \cdot y \in L_n$. Notice that the only string of length n that is not a member of S_x is the string that differs from x at every bit position. Therefore, every string x of length n defines a unique set S_x. It follows that there are 2^n distinct sets S_x.

Now, consider a DFA that accepts L_n. This DFA must contain at least one state for each possible set S_x. As a result, the DFA must have at least 2^n states. ◀

2.4 Sparsely Balanced Automata

A tree shaped automaton is said to be **sparsely balanced** if (1) the final states are exactly the leaves and (2) every branch has the same length. As a result, all strings accepted by a sparsely balanced tree shaped automaton have the same length. We call this length the **height** of the automaton and denote the height by h.

It's worth noticing that every sparsely balanced tree shaped automaton over a binary input alphabet has a corresponding disjunctive normal form (DNF) formula such that the strings accepted by the automaton are exactly the satisfying assignments for the formula.

Let a sparsely balanced tree shaped automaton D of height h with branches $\{b_i\}_{i\in[m]}$ be given. The corresponding DNF formula ϕ has h variables and is expressed as $\vee_{i\in[m]}C_i$ where each C_i is just a conjunction of literals. Further, each level of D is represented by a variable of ϕ and each branch b_i is represented by a conjunction of literals C_i satisfying that for each variable v_j, either (1) C_i contains v_j and the jth position of b_i has a 0 transition, (2) C_i contains $\bar{v_j}$ and the jth position of b_i has a 1 transition, or (3) C_i doesn't contain v_j or $\bar{v_j}$ and the jth position of b_i contains a multi-label 0 or 1 transition.

Although many of the tree shaped automata that appear in our later constructions will be sparsely balanced, we will not specifically focus on the sparsely balanced property.

3 Fine-Grained Complexity

3.1 NP-Completeness

We present reductions from boolean satisfiabilty to intersection non-emptiness. In particular, we focus on intersection non-emptiness for tree shaped DFA's over a binary input alphabet. That is, given a finite list of tree shaped DFA's over a binary input alphabet, does there exist a bit string that simultaneously satisfies all of the DFA's? We abbreviate this problem by $\text{IE}_{\mathcal{TD}}$.

▶ **Proposition 4.** $\text{IE}_{\mathcal{TD}}$ *is* NP*-complete. Moreover, the problem is still* NP*-complete when each DFA has at most 3 branches.*

Proof. A witness is any string that is in the intersection. Since the witness length is linear in the number of states and verification just consists of running the DFA's, $\text{IE}_{\mathcal{TD}}$ is in NP.

Hardness follows by a reduction from 3-SAT similar to that found in [19]. For a given 3-SAT formula $\phi = \bigwedge_{i\in[m]} C_i$ with n variables and m clauses, we construct a tree shaped DFA for each clause. In particular, for each $i \in [m]$, we build a DFA D_i that checks if a given bit assignment satisfies C_i. Each DFA has three branches where each branch represents a possible way for the clause to be satisfied.

- Branch 1: first literal is true.
- Branch 2: first literal is false and second is true.
- Branch 3: first and second literals are false and third is true.

The construction yields m tree shaped DFA's each with $O(n)$ states. The DFA's all have at most 3 branches and collectively verify that a given bit string represents a satisfying assignment of ϕ. ◀

It's worth noting that if each DFA has at most two branches, then the intersection problem reduces to 2-SAT and therefore is solvable in polynomial time.

3.2 ETH-Hardness

The exponential time hypothesis (ETH) states that 3-SAT is not solvable in $2^{o(n)}$ time. It has been shown that k-clique is hard for ETH in the sense that if k-clique $\in$ DTIME($n^{o(k)}$), then ETH is false [6]. Similarly, we will show that the intersection non-emptiness problem for k tree shaped DFA's over a binary input alphabet is ETH-hard. We abbreviate this intersection problem by k-IE$_{\mathcal{TD}}$.

When discussing SAT problems, we use n to denote the number of variables and m to denote the number of clauses. The next lemma follows implicitly from the Sparsification Lemma [8]. Sparsification allows one to convert a single SAT instance to many SAT instances with fewer clauses.

▶ **Lemma 5.** *If 3-SAT is solvable in $2^{o(m)}$ time, then ETH is false.*

Sketch of proof. Suppose that 3-SAT is solvable in $2^{o(m)}$ time where m represents the number of clauses. That is, for every $\varepsilon > 0$ we can solve 3-SAT in $O(2^{\epsilon m})$ time. By the Sparsification Lemma [8], it follows that for every $\varepsilon > 0$ we can solve 3-SAT in $O(2^{\epsilon n})$ time where n denotes the number of variables. Further, by definition, we get that ETH is false. ◀

▶ **Theorem 6.** *If k-IE$_{\mathcal{TD}}$ is solvable in $n^{o(k)}$ time, then ETH is false.*

Proof. Suppose that k-IE$_{\mathcal{TD}}$ is solvable in $n^{o(k)}$ time. That is, for every $\varepsilon > 0$ there exists k sufficiently large such k-IE$_{\mathcal{TD}} \in$ DTIME($n^{\varepsilon k}$). Now, we show that 3-SAT is solvable in $2^{o(m)}$ time.

Let $\varepsilon > 0$ be given. Let a 3-CNF formula ϕ with n variables and m clauses be given. Using the construction from the proof of Proposition 4, we can build m tree shaped DFA's of height n with 3 branches each such that ϕ is satisfiable if and only if the DFA's have a non-empty intersection.

By the assumption, we can pick $k \in \mathbb{N}$ such that k-IE$_{\mathcal{TD}} \in$ DTIME($n^{\frac{\varepsilon k}{\log_2(3)}}$). Break the m DFA's up into k groups of $\frac{m}{k}$ each. Take the pruned product of all of the DFA's in each of the groups to get k larger DFA's. By Lemma 2, each of the pruned product DFA's is a tree shaped DFA with at most $3^{\frac{m}{k}}$ states of height n. Further, since each pruned product forms a tree of height n, each pruned product has at most $n \cdot 3^{\frac{m}{k}}$ states total. Applying the efficient algorithm for k-IE$_{\mathcal{TD}}$, we can solve the intersection problem for these k DFA's in $O(n^{\frac{\varepsilon k}{\log_2(3)}} \cdot 2^{\varepsilon m})$ time. Therefore, 3-SAT is solvable in $O(n^{\frac{\varepsilon k}{\log_2(3)}} \cdot 2^{\varepsilon m})$ time.

Since ε was arbitrary, it follows that 3-SAT is solvable in $2^{o(m)}$ time. By Lemma 5, it follows that ETH is false. ◀

3.3 SETH-Hardness

We denote by CNF-SAT the classic satisfiability problem for formulas in conjunctive normal form. Further, we denote by c-SAT the satisfiability problem for formulas in conjunctive normal form with at most c literals per clause.

The strong exponential time hypothesis (SETH) states that there is no $\varepsilon > 0$ such that for every c, we have c-SAT is solvable in $O(2^{(1-\varepsilon)n})$ time where n denotes the number of variables. It has been shown that the orthogonal vectors problem is hard for SETH in the sense that if it is solvable in $O(n^{2-\varepsilon})$ time, then SETH is false [24, 25]. Similarly, we will show that intersection non-emptiness for two tree shaped DFA's is SETH-hard.

▶ **Theorem 7.** *If 2-IE$_{\mathcal{TD}}$ is solvable in $O(n^{2-\varepsilon})$ time, then SETH is false.*

Proof. We define a special kind of reduction from CNF-SAT to 2-IE$_{\mathcal{TD}}$. Let a CNF formula ϕ with n variables and m clauses be given. We construct DFA's D_1 and D_2 with $(m+n)\cdot 2^{\frac{n}{2}}$ states each such that ϕ is satisfiable if and only if $L(D_1)\cap L(D_2)$ is non-empty.

A variable assignment for ϕ is a bit string of length n. Consider an assignment α. We can break α up into two blocks α_1 and α_2 of $\frac{n}{2}$ bits each. Additionally, we consider a clause assignment for α. A clause assignment is a bit string of length m. We say that a clause assignment is valid for α if for each clause c_i of ϕ, at least one of the following is satisfied:

- the ith bit of the clause assignment is 0 and α_1 forces c_i to be satisfied.
- the ith bit of the clause assignment is 1 and α_2 forces c_i to be satisfied.

The DFA's D_1 and D_2 will read as input a variable assignment α consisting of blocks α_1 and α_2 followed by a corresponding clause assignment β. The DFA D_1 branches while reading α_1 and ignores α_2. Similarly, D_2 ignores α_1 and branches while reading α_2. Then, D_1 reads the clause assignment β and verifies that for each i, if α_1 doesn't force clause c_i to be satisfied, then β has a 1 at the ith position. Similarly, D_2 reads the clause assignment β and verifies that for each i, if α_2 doesn't force clause c_i to be satisfied, then β has a 0 at the ith position.

Together, D_1 and D_2 collectively verify that for each clause c_i, either α_1 or α_2 forces c_i to be satisfied. Therefore, D_1 and D_2 collectively verify that β is valid for α. Further, ϕ is satisfiable if and only if there is a variable assignment α and clause assignment β such that β is valid for α. We conclude that ϕ is satisfiable if and only if D_1 and D_2 have a non-empty intersection. ◀

It has been shown that for every fixed k, k-dominating set is hard for SETH in the sense that if it is solvable in $O(n^{k-\varepsilon})$ time for some $\varepsilon > 0$ and $k \geq 3$, then SETH is false [17]. Similarly, we will show that intersection non-emptiness for k tree shaped DFA's over the input alphabet $[k]$ is SETH-hard. We abbreviate this intersection problem by (k,k)-IE$_{\mathcal{TD}}$.

▶ **Theorem 8.** *If there exists k such that (k,k)-IE$_{\mathcal{TD}}$ is solvable in $O(n^{k-\varepsilon})$ time, then SETH is false.*

Proof. In a similar manner as Theorem 7, we reduce CNF-SAT to the intersection problem. Let a natural number k be given. Let a CNF formula ϕ with n variables and m clauses be given. We construct DFA's $\{D_i\}_{i\in[k]}$ with $(m+n)\cdot 2^{\frac{n}{k}}$ states each such that ϕ is satisfiable if and only if $\bigcap_{i\in[k]} L(D_i)$ is non-empty.

Each variable assignment α for ϕ is broken up into k blocks $\{\alpha_i\}_{i\in[k]}$ of $\frac{n}{k}$ bits each. A clause assignment for α is a string of m characters over the alphabet $[k]$. A clause assignment is valid for α if for each clause, if the corresponding character is $i\in[k]$, then α_i forces the clause to be satisfied.

The DFA's each read in an assignment α followed by a clause assignment β. The DFA D_i verifies that if the ith block doesn't satisfy a clause, then the corresponding clause assignment character is not i. Collectively, the DFA's verify that β is valid for α. As a result, we get that ϕ is satisfiable if and only if the DFA's have a non-empty intersection. ◀

3.4 c-SAT Hardness

We write (c,k)-IE$_{\mathcal{TD}}$ to denote intersection non-emptiness for k tree shaped DFA's over the input alphabet $[c]$. As the parameter c gets larger, the intersection problem gets harder. In particular, we show that for each c, faster algorithms for the intersection problem over the input alphabet $[c]$ implies faster algorithms for c-SAT.

▶ **Theorem 9.** *If (c,k)-IE$_{\mathcal{TD}}$ is solvable in $O(n^\delta)$ time, then c-SAT is solvable in $O(2^{\frac{\delta n}{k}})$ time.*

Sketch of proof. In Theorem 8, it was shown how to reduce CNF-SAT to (k,k)-IE$_{\mathcal{BT}}$. If we instead consider c-SAT, we can do a similar reduction where the DFA's are over the alphabet $[c]$. Since each clause has at most c literals, at most c assignment blocks are relevant for determining if that clause is satisfied. For each clause, a character from $[c]$ will be used to represent which of the relevant blocks forces the clause to be satisfied. ◀

4 Parameterized Reductions

4.1 Definitions

We present parameterized reductions between several related problems. All of these reductions will be specific kinds of fpt-reductions that we refer to as LBL-reductions.

Any parameterized problem can be represented by an infinite family of fixed levels. Let families $\{k\text{-}\mathcal{A}\}_{k\in\mathbb{N}}$ and $\{k\text{-}\mathcal{B}\}_{k\in\mathbb{N}}$ be given. We say that k-$\mathcal{A}$ is **fpt-reducible** to k-$\mathcal{B}$ if there exists a family of reduction functions $\{r_k\}_{k\in\mathbb{N}}$ and functions f, g such that (1) for all $k\in\mathbb{N}$ and all instances x of k-$\mathcal{A}$, we have $x\in k\text{-}\mathcal{A} \iff r_k(x)\in f(k)\text{-}\mathcal{B}$, and (2) there exists a constant c such that for all $k\in\mathbb{N}$, r_k is computable in $g(k)\cdot n^c$ time. Further, we say that the ftp-reduction is **uniform** if the family of reduction functions is effectively computable. In parameterized complexity, uniform ftp-reduction is the most standard and commonly used notion of reduction [7].

An **LBL-reduction** is a special kind of fpt-reduction such that for all $k\in\mathbb{N}$, $f(k)=k$ [20]. This means that the reductions exactly preserve the parameter[5]. Further, we say that the **instance blow-up** of an LBL-reduction is $O(n^d)$ if for every k, an instance x of size n is reduced to an instance $r_k(x)$ of size at most $O(n^d)$.

4.2 Clique

Consider the intersection non-emptiness problem for k tree shaped NFA's over a binary input alphabet. We abbreviate this problem by k-IE$_{\mathcal{TN}}$.

It has been shown that by using fast matrix multiplication, we can solve k-clique in $O(n^{0.792k})$ time for graphs with n vertices and up to n^2 edges [21]. Further, by reducing k-IE$_{\mathcal{TN}}$ to k-clique, we get an $O(n^{0.792k})$ time algorithm for k-IE$_{\mathcal{TN}}$.

▶ **Theorem 10.** *k-IE$_{\mathcal{TN}}$ is LBL-reducible to k-clique with $O(n^2)$ instance blow-up.*

Proof. Let a list of tree shaped NFA's $\{N_i\}_{i\in[k]}$ be given. Each NFA has at most m branches. From all the NFA's combined, there are at most $k\cdot m$ branches. We construct a graph G where each vertex represents a branch. As a result, G has $k\cdot m$ vertices.

Consider branches b_i and b_j. We say that b_i and b_j have a bit mismatch if there is some character position where b_i has a 0 bit and b_j has a 1 bit or vice versa. Now, there is an edge in G between b_i and b_j if the following are satisfied: b_i and b_j (1) have the same length, (2) come from different NFA's, and (3) have no bit mismatches.

We argue that the NFA's have a non-empty intersection if and only if G has a k-clique. First, if the NFA's have a non-empty intersection, then there is a string that is accepted by each of the k NFA's. Further, the string determines a branch through each of the NFA's

[5] The notion of a parameter-preserving reduction from [12] is weaker only requiring that $f(k)=\mathsf{poly}(k)$.

where the resulting branches contain no bit mismatches. As a result, the k branches form a k-clique in G. Second, if G has a k-clique, then there are k branches (one for each NFA) such that there are no bit mismatches. Therefore, for each bit position, either a 0 or 1 bit is consistent with all k branches. As a result, we can select bits to form a string that is consistent with the k branches. This bit string will satisfy all of the NFA's. ◀

In the preceding reduction, given an instance of k-IE$_{\mathcal{TN}}$ with total size n, we construct a k-clique instance with at most n vertices and n^2 edges.

▶ **Corollary 11.** *For all $k \geq 3$, k-IE$_{\mathcal{TN}}$ is solvable in $O(n^{0.792k})$ time.*

4.3 Multi-Parameter Equivalences

We show that for all fixed c, the following three problems are LBL-equivalent to (c,k)-IE$_{\mathcal{TD}}$.

- Intersection non-emptiness problem for k tree shaped NFA's over the input alphabet $[c]$. We abbreviate this problem by (c,k)-IE$_{\mathcal{TN}}$.
- c-Uniform k-hyperclique: given a c-uniform hypergraph H, does there exist a complete hypergraph with k vertices in H?
- k-Weighted c-CNF satisfiability: given a c-CNF boolean formula ϕ, does there exist a satisfying assignment for ϕ with exactly k ones?

First, by observing that every DFA is an NFA, we get the following proposition.

▶ **Proposition 12.** *For all fixed c, (c,k)-IE$_{\mathcal{TD}}$ is LBL-reducible to (c,k)-IE$_{\mathcal{TN}}$ with $O(n)$ instance blow-up.*

Next, we reduce intersection non-emptiness for tree shaped NFA's to the clique problem.

▶ **Theorem 13.** *For all fixed c, (c,k)-IE$_{\mathcal{TN}}$ is LBL-reducible to c-uniform k-hyperclique with $O(n^c)$ instance blow-up.*

Proof. Let a list of tree shaped NFA's $\{N_i\}_{i\in[k]}$ over input alphabet $[c]$ be given. Each NFA has at most m branches. From all the NFA's combined, there are at most $k \cdot m$ branches. We construct a c-uniform graph H where each vertex represents a branch. As a result, H has $k \cdot m$ vertices.

Consider a set of c branches. For each branch and character position, we get a set of possible characters from $[c]$. In other words, for each character position, we get c subsets of $[c]$ where each subset is associated with one of the branches. Now, there is a hyperedge in H between a set of c branches if the following are satisfied: (1) the c branches all have the same length, (2) no two branches come from the same NFA, and (3) for each position, the corresponding c subsets of $[c]$ have a non-empty intersection.

We claim that the NFA's have a non-empty intersection if and only if H has a k-hyperclique. First, if the NFA's have a non-empty intersection, then there exists a string that satisfies all of the NFA's. This string determines a branch for each of the NFA's where the branches form a k-hyperclique in H. Second, if there exists a k-hyperclique, then there are k branches such that for every choice of c branches and every character position, the corresponding c subsets of $[c]$ have a non-empty intersection. Therefore, for every character position, the corresponding k subsets of $[c]$ must also have a non-empty intersection or else we would be able to pick c of the k subsets to get an empty intersection. As a result, for each position, we can pick a character in the non-empty intersection. These choices of characters form a string that is accepted by each of the NFA's. ◀

Now, we reduce the clique problem to weighted CNF satisfiability.

▶ **Theorem 14.** *For all fixed c, c-uniform k-hyperclique is LBL-reducible to k-weighted c-CNF satisfiability with $O(n^c)$ instance blow-up.*

Proof. Let a c-uniform hypergraph H with vertices $\{v_i\}_{[n]}$ be given. We construct a c-CNF formula ϕ with n corresponding variables $\{v_i\}_{[n]}$ and up to $O(n^c)$ clauses. For each set S of c vertices from H such that S does not form a hyperedge of H, ϕ has a corresponding clause. This clause consists of literals $\bar{v_i}$ such that $v_i \in S$. As a result, the clause fails to be satisfied exactly when the c variables are assigned the value 1. Now, it remains to show that H has a k-hyperclique if and only if ϕ has a weight k satisfying assignment.

First, if H has a k-hyperclique, then we can assign the corresponding k variables the value 1 and the remaining $n-k$ variables the value 0. Consider an arbitrary clause C_i of ϕ. The weight k assignment must satisfy C_i or else we would get that the corresponding c vertices do not form a hyperedge of H yet are still members of the k-hyperclique which is a contradiction. Second, if ϕ has a weight k satisfying assignment, then we consider the corresponding set S of k vertices. Since all of the clauses were satisfied, none of the clauses could have only contained variables corresponding to vertices from S. Therefore, every subset of c vertices must form a hyperedge of H. ◀

Finally, we reduce weighted CNF satisfiability to intersection non-emptiness for tree shaped DFA's.

▶ **Theorem 15.** *For all fixed c, k-weighted c-CNF satisfiability is LBL-reducible to (c,k)-$\mathrm{IE}_{\mathcal{TD}}$ with $O(n^3)$ instance blow-up.*

Proof. The reduction is related to the reductions from Theorem 8 and 9. Let a c-CNF formula of size n be given. We construct k tree shaped DFA's each with at most n^3 states such that the formula has a k-weighted satisfying assignment if and only if the DFA's have a non-empty intersection.

The DFA reads an encoded k-weighted variable assignment followed by a clause assignment. The k-weighted variable assignment is encoded as a sequence of k bit strings each of length $\log(n)$. Each bit strings represents one of the k variables that are assigned the value 1. Then, the clause assignment is a sequence of characters from $[c]$ where the ith character in the sequence represents a choice of one variable from the c variables in the ith clause.

Let $i \in [k]$ be given. The ith DFA will only branch based on the ith and $(i+1)$th bit strings from the variable assignment. This will lead to n^2 branches each of length up to n for reading the clause assignment. The DFA will verify that when bit strings are interpreted as numerical values, the ith bit string is less than the $(i+1)$th bit string. The branching can then be interpreted as storing the ith variable that is assigned the value 1 followed by all variables assigned the value 0 up until the $(i+1)$th variable that is assigned the value 1. As a result, this DFA now has one of k blocks that make up the variable assignment.

Next, the DFA reads the clause assignment and will only pay attention to characters for clauses that the block fails to satisfy. For such clauses, the DFA will make sure that none of that blocks characters are read. Since the alphabet is $[c]$, this requires no branching.

Together, the DFA's each branch based on a block of the assignment and together verify that we never have all k blocks failing to satisfy a clause. As a result, we get that their intersection is non-empty if and only if there exists a satisfying assignment. ◀

By combining the reductions carefully, we can guarantee that we have at most an $O(n^{3c})$ instance blow-up between any two of the problems.

▶ **Corollary 16.** *For all fixed c, the following are LBL-equivalent with $O(n^{3c})$ instance blow-up: (c,k)-$\mathrm{IE}_{\mathcal{TD}}$, (c,k)-$\mathrm{IE}_{\mathcal{TN}}$, c-uniform k-hyperclique, and k-weighted c-CNF satisfiability.*

It is known that for all fixed $c \geq 2$, weighted c-CNF satisfiability is $W[1]$-complete [5]. Further, by the equivalences from Corollary 16, we observe that for all fixed $c \geq 2$, (c,k)-IE$_{\mathcal{TD}}$, (c,k)-IE$_{\mathcal{TN}}$, and c-uniform k-hyperclique are also $W[1]$-complete. Although, the fine-grained analysis of these problems is much more telling than the observation of their $W[1]$-completeness because the typical notion of fpt-reduction is quite weak in comparison by failing to preserve the parameter and not including details on the instance size blow-up.

5 Conclusion

We showed that not only is intersection non-emptiness for tree shaped automata NP-complete, but it is also both ETH-hard and SETH-hard. In particular, in Theorem 6, we showed that if k-IE$_{\mathcal{TD}}$ is solvable in $O(n^{o(k)})$ time, then ETH is false. Then, in Theorem 7, we showed that if 2-IE$_{\mathcal{TD}}$ is solvable in $O(n^{2-\varepsilon})$ time, then SETH is false.

Further, we investigated the complexity of intersection non-emptiness for tree shaped automata over larger input alphabets. In Theorem 8, we showed that if there exists k such that (k,k)-IE$_{\mathcal{TD}}$ is solvable in $O(n^{k-\varepsilon})$ time, then SETH is false. Finally, in Section 4, we presented LBL-reductions to show that for every fixed c, intersection non-emptiness for k tree shaped automata over the input alphabet $[c]$ is equivalent to k-weighted satisfiability for c-CNF formulas and k-clique for c-uniform hypergraphs.

Although we were able to show that $(2,k)$-IE$_{\mathcal{TD}}$ is solvable in $O(n^{k-\varepsilon})$ time in Corollary 11, we were unable to show that there exists k such that $(3,k)$-IE$_{\mathcal{TD}}$ is solvable in $O(n^{k-\varepsilon})$ time. As a result, we suggest that one may be able to prove more conditional lower bounds for the (c,k)-IE$_{\mathcal{TD}}$ problems.

Further, the faster algorithm for $(2,k)$-IE$_{\mathcal{TD}}$ breaks down if we allow one of the DFA's to be acyclic. Consider intersection non-emptiness for one acyclic DFA and $k-1$ tree shaped DFA's over a binary input alphabet. We can carry out a variation of the reduction from Theorem 8 to show that the existence of an $O(n^{k-\varepsilon})$ time algorithm would imply that SETH is false. In particular, instead of assigning each clause a character from $[k]$, we would assign each clause a bit string of length up to k. Such a bit string would represent which blocks force the clause to be satisfied and which blocks don't.

Acknowledgments.

I greatly appreciate all of the help and suggestions that I received. In particular, I appreciate the helpful ideas and suggestions shared by Michael Blondin, Noy Rotbart, Joseph Swernofsky, Ryan Williams, and Nils Wisiol. Also, I appreciate the discussion and comments from the cstheory stackexchange community on "DFA intersection algorithm for special cases" and "Deciding emptiness of intersection of regular languages in subquadratic time". Further, I thank Richard Lipton, Kenneth Regan, and all those from University at Buffalo who continue to offer kind comments and encouragement.

References

1 N. Alon and Wolfgang Maass. Meanders, ramsey theory and lower bounds for branching programs. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 410–417, Oct 1986.

2 Michael Blondin. Complexité du problème d'intersection d'automates. B.sc. honour thesis, Université de Montréal, 2009.

3 Michael Blondin. Complexité raffinée du problème d'intersection d'automates. M.sc. thesis, Université de Montréal, 2012.

4 Michael Blondin, Andreas Krebs, and Pierre McKenzie. The complexity of intersecting finite automata having few final states. *Computational Complexity*, 2014.

5 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized np-hard problems. *Information and Computation*, 201(2):216 – 231, 2005.

6 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346 – 1367, 2006.

7 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

8 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512 – 530, 2001.

9 G. Karakostas, R. J. Lipton, and A. Viglas. On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, 302:257–274, 2003.

10 Takumi Kasai and Shigeki Iwata. Gradually intractable problems and nondeterministic log-space lower bounds. *Mathematical systems theory*, 18(1):153–170, 1985.

11 Dexter Kozen. Lower bounds for natural proof systems. *Proc. 18th Symp. on the Foundations of Computer Science*, pages 254–266, 1977.

12 Philipp Kuinke. Survey of parameter-preserving reductions. Bachelor thesis, RWTH Aachen University, 2013.

13 Klaus-Jörn Lange and Peter Rossmanith. The emptiness problem for intersections of regular languages. *Lecture Notes in Computer Science*, 629:346–354, 1992.

14 R. J. Lipton. On the intersection of finite automata. *Gödel's Lost Letter and P=NP*, August 2009.

15 R. J. Lipton and K. W. Regan. The power of guessing. *Gödel's Lost Letter and P=NP*, November 2012.

16 Bernard M. E. Moret. Decision trees and diagrams. *ACM Comput. Surv.*, 14(4):593–623, December 1982.

17 Mihai Pătraşcu and Ryan Williams. On the possibility of faster sat algorithms. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1065–1075, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

18 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal*, 1959.

19 Narad Rampersad and Jeffrey Shallit. Detecting patterns in finite regular and context-free languages. *Information Processing Letters*, 110, 2010.

20 Joseph Swernofsky and Michael Wehar. On the complexity of intersecting regular, context-free, and tree languages. In *ICALP 2015 (Part II)*, pages 414–426, 2015.

21 Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254 – 257, 2009.

22 Michael Wehar. Intersection emptiness for finite automata. Honors thesis, Carnegie Mellon University, 2012.

23 Michael Wehar. Hardness results for intersection non-emptiness. In *ICALP 2014 (Part II)*, pages 354–362, 2014.

24 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2–3):357 – 365, 2005.

25 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1867–1877. SIAM, 2014.