On the Complexity of Intersection Non-Emptiness Problems

by

Michael Wehar

December 1, 2016

A dissertation submitted to the Faculty of the Graduate School of
the University at Buffalo, State University of New York
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Engineering

Acknowledgements

I wouldn't have been able to complete this work without the loving support of my family, girlfriend, and friends. In addition, I'm grateful for the guidance of my advisor Kenneth Regan and the tremendous community at University at Buffalo.

I would like to thank a few especially supportive collaborators, friends, and acquaintances whose kindness and encouragement significantly helped me throughout my research pursuits. Thank you Michael Blondin, Dmitry Chistikov, James Clay, Chaowen Guan, Andrew Hughes, Joseph Swernofsky, and Chen Xu. In addition, I would like to thank my thesis committee including Atri Rudra and Hung Ngo, all of my collaborators, and all those from Carnegie Mellon University who supported me on an honors thesis of the same topic.

Contents

Abstract

A central problem in formal language theory is deciding whether a finite list of regular languages has a non-empty intersection. That is, given a finite list of DFA's (deterministic finite automata), does there exist a string that satisfies all of the DFA's? This problem is known as the intersection non-emptiness problem for finite automata.

Intersection non-emptiness can be viewed as a constraint satisfaction problem. In particular, we can view each automaton as verifying some constraint. Then, determining if there exists a string that satisfies all of the automata is equivalent to determining if there exists a string that satisfies all of the constraints. Also, intersection non-emptiness can be viewed as a graph reachability problem. In particular, we can consider the Cartesian product of all of the automata. Then, determining if there exists a string that satisfies all of the automata is equivalent to determining if there exists a directed path from the start state to a final state in the product automaton.

The dual nature of intersection non-emptiness as both a constraint satisfaction problem and a graph reachability problem allows us to build parameterized reductions from fundamentally hard problems to intersection non-emptiness. These parameterized reductions allow us to characterize classical time and space complexity classes. In this thesis, we examine intersection non-emptiness problems for various kinds of automata in an effort to comprehensively characterize the classical time and space complexity classes.

1

Introduction

1.1 Formal Statement

We investigate problems where we are given an encoding of a finite list of automata and want to determine whether there exists a string that satisfies each automaton in the list. A problem of this form is referred to as an intersection non-emptiness problem (IE) because it is equivalent to determining whether the languages associated with the automata have a non-empty intersection.

Consider the following classical intersection non-emptiness problem for deterministic finite automata. Given a finite list of DFA's, does there exist a string that simultaneously satisfies all of the DFA's? This problem is equivalent to determining if the regular languages corresponding to the DFA's have a non-empty intersection.

1.2 Motivation

Intersection non-emptiness is motivated as a constraint satisfaction problem and a graph reachability problem.

Constraint Satisfaction: To determine whether a mathematical object exists one could start by listing constraints for the proposed object to satisfy. Then, for each constraint one could build a verifier to computationally determine whether an input satisfies the constraint. If each constraint can be verified by an automaton,

does that mean one could efficiently determine whether there exists an object that satisfies all of the constraints?

Graph Reachability: The Cartesian product construction is a classical construction that is used for showing that regular languages are closed under intersection [55]. The idea is that given two DFA's D_1 and D_2, there exists a product DFA $\mathcal{D}$ such that $L(\mathcal{D}) = L(D_1) \cap L(D_2)$. Therefore, determining whether D_1 and D_2 have a non-empty intersection is equivalent to determining if the product has a non-empty language. In other words, D_1 and D_2 have a non-empty intersection if and only if there exists a path in the product automaton's state diagram from the start state to a final state.

Hardness: We assert that intersection non-emptiness is a fundamentally hard problem. We justify this assertion by exploiting the dual nature of intersection non-emptiness as a constraint satisfaction problem and a graph reachability problem to build reductions from Turing machine acceptance problems to intersection non-emptiness problems. In particular, we construct finite automata that collectively verify whether an input string is accepted by a Turing machine. We express the Turing machine acceptance as graph reachability and break the reachability into distinct constraints that can be verified by distinct automata. Using this approach, we are able to characterize classical time and space complexity classes.

1.3 History

1.3.1 Intersection Non-Emptiness

Kozen (1977) showed that the intersection non-emptiness problem for DFA's (deterministic finite automata) is PSPACE-complete [39]. Kasai and Iwata (1985) showed that solving intersection non-emptiness for k DFA's is equivalent to simulating a $k\log(n)$-space bounded non-deterministic Turing machine with a binary work tape alphabet [37]. Next, Lange and Rossmanith (1992) showed that if we constrain k (the number of DFA's) to be $\log(n)$, then $\text{IE}_{\mathcal{D}}$ becomes $\text{NSPACE}(\log^2(n))$-complete [45]. Then, Karakostas, Lipton, and Viglas (2003) connected the hardness

of intersection non-emptiness to the hardness of Boolean Satisfiability [36]. They showed that if the parameterized intersection problem for k DFA's is solvable in $n^{o(k)}$ time, then the exponential time hypothesis is false[1]. Further, they investigated a variation of the problem with $k+1$ DFA's where one DFA is much larger than the other k DFA's. They denoted the size of the larger DFA by m and the size of the largest of the smaller DFA's by n. They showed that if this version of intersection non-emptiness is solvable in $m \cdot n^{o(k)}$ time, then NL $\subseteq$ DTIME($n^{1+\varepsilon}$) for all $\varepsilon > 0$.

1.3.2 Related Problems

Hardness results have been established for intersection non-emptiness over restricted classes of finite automata. Finite automata are naturally restricted either by algebraic properties or graphical properties.

On algebraic restrictions, Blondin (2009, 2012, 2014) extensively surveyed work on the intersection non-emptiness problems for automata with restricted transition monoids [4, 5, 6]. However, we focus on graphical restrictions. Rampersad and Shallit (2010) showed that intersection non-emptiness for DFA's that accept finite regular languages is NP-complete [56]. All such DFA's can be represented by acyclic DFA's. Therefore, the intersection non-emptiness problem for acyclic DFA's is NP-complete. Wareham (2001) explored the complexity of an equivalent problem referred to as the bounded DFA intersection problem. Their results on parameterizing based on the number of automata imply that intersection non-emptiness for k acyclic DFA's is $W[t]$-hard for all $t \geq 1$ [61]. An upper bound of Cesati (2003) implies that intersection non-emptiness for k-acyclic DFA's is in $W[P]$ [9].

Deterministic finite automata over a unary alphabet graphically take the form of a pan consisting of a handle and a cycle. Such automata have also been referred to as Tally-DFA's. Following from results of Meyer and Stockmeyer (1973) and Galil (1976), intersection non-emptiness for unary DFA's is NP-complete [58, 29, 45]. Further, Fernau and Krebs (2016) considered parameterizations of the intersection

[1]Recent work by Fernau and Krebs (2016) expands on results of this form [26].

non-emptiness problem for unary DFA's by the number of DFA's and the number of states. They showed that the existence of faster parameterized algorithms implies that ETH (exponential time hypothesis) is false [26].

Tree automata are a generalization of string-based automata that read labeled trees as input in a top-down or bottom-up fasion [17]. Veanes (1997) showed that intersection non-emptiness problem for tree automata is EXP-complete [17].

2

Preliminaries

2.1 Turing Machines

Whenever we use the term Turing machine, we refer to a deterministic, non-deterministic, symmetric, or alternating tape machine. We particularly focus on what we refer to as a binary Turing machine. Such a machine has a two-way read only input tape and a two-way read/write work tape that is restricted to a binary alphabet. A work tape over a binary alphabet will be referred to as a binary work tape. A cell on a binary work tape will be referred to as a bit cell.

A Turing machine M is said to accept a language L if for all input strings x, M accepts x if and only if $x \in$ L. Further, M is said to decide L if M also halts on all inputs. For deterministic machines, M is said to be $f(n)$-time bounded if for all input strings x, M runs for at most $f(n)$ steps on input x where n denotes the length of x. Further, M is said to be $f(n)$-space bounded if for all input strings x, M accesses at most $f(n)$ distinct work tape cells on input x where n denotes the length of x.

Following the convention of [11, 43], for non-deterministic and alternating machines, M is said to be $f(n)$-time bounded if for all accepted input strings x, there exists an accepting computation of height at most $f(n)$. Further, M is said to be $f(n)$-space bounded if for all accepted input strings x, there exists an accepting computation where every configuration is bounded to $f(n)$ work tape cells.

2.2 Complexity

2.2.1 Complexity Measures

Complexity is typically measured relative to a resource for a class of computational machines. For example, we measure the time complexity for deterministic Turing machines and denote this measure by DTIME. In addition to DTIME, we have the following time complexity measures.

NTIME : Time for non-deterministic Turing machines

ATIME : Time for alternating Turing machines

STIME : Time for symmetric Turing machines

AuxTIME : Time for deterministic auxiliary pushdown automata

AltAuxTIME : Time for alternating auxiliary pushdown automata

Also, we measure the space complexity for deterministic Turing machines and denote this measure by DSPACE. In addition to DSPACE, we have the following space complexity measures.

NSPACE : Space for non-deterministic Turing machines

ASPACE : Space for alternating Turing machines

SSPACE : Space for symmetric Turing machines

AuxSPACE : Space for deterministic auxiliary pushdown automata

AltAuxSPACE : Space for alternating auxiliary pushdown automata

In some cases, we need to measure the time or space complexity for tape machines with a restricted number of tapes and alphabet size. For example, when

measuring complexity relative to a binary Turing machine, that is a tape machine with a read only input tape and a single binary work tape, we use a superscript b. In particular, the complexity measure NSPACE^b measures space complexity for non-deterministic binary Turing machines.

2.2.2 Complexity Classes

A complexity class is a set of languages. We typically define such a set of languages by a class of resource bounded computational machines. We can specify such a class of machines using both a complexity measure and a bound.

For example, consider the complexity class $\text{DTIME}(f(n))$. This set of languages is defined by the class of $f(n)$-time bounded deterministic Turing machines. In particular, $\text{L} \in \text{DTIME}(f(n))$ if and only if there exists a deterministic $f(n)$-time bounded Turing machine M such that M decides L. Similarly, consider the complexity class $\text{DSPACE}(f(n))$. This set of languages is defined by the class of $f(n)$-space bounded deterministic Turing machines. We write $\text{L} \in \text{DSPACE}(f(n))$ if and only if there exists a deterministic $f(n)$-space bounded Turing machine M such that M decides L.

Consider the following list of complexity classes and their corresponding classes

of resource bounded computational machines.

L	:	Logarithmic space bounded deterministic Turing machines
SL	:	Logarithmic space bounded symmetric Turing machines
NL	:	Logarithmic space bounded non-deterministic Turing machines
AL	:	Logarithmic space bounded alternating Turing machines
AuxL	:	Logarithmic space bounded deterministic auxiliary pushdown automata
P	:	Polynomial time bounded deterministic Turing machines
AltAuxL	:	Logarithmic space bounded alternating auxiliary pushdown automata
EXP	:	Exponential time bounded deterministic Turing machines

2.2.3 Acceptance Problems

The general form of an acceptance problem is as follows. Given an encoding[1] of a machine M and an input x, does M accept x? For each k and each machine class from the preceding subsection, we can define an acceptance problem. Consider the

[1] A time or space bounded Turing machine is encoded by the combination of a Turing machine encoding and a resource bound. The machine rejects if it happens to try to use more resources than the bound allows.

following acceptance problems and their associated machine classes.

$$
\begin{aligned}
D^S_{k\log} &: k\log(n)\text{-space bounded deterministic binary Turing machines}\\
S^S_{k\log} &: k\log(n)\text{-space bounded symmetric binary Turing machines}\\
N^S_{k\log} &: k\log(n)\text{-space bounded non-deterministic binary Turing machines}\\
A^S_{k\log} &: k\log(n)\text{-space bounded alternating binary Turing machines}\\
Aux^S_{k\log} &: k\log(n)\text{-space bounded auxiliary binary pushdown automata}\\
D^T_{n^k} &: n^k\text{-time bounded deterministic binary Turing machines}\\
AltAux^T_{k\log} &: k\log(n)\text{-space bounded alternating auxiliary binary pushdown automata}\\
D^T_{2^{n^k}} &: 2^{n^k}\text{-time bounded deterministic binary Turing machines}
\end{aligned}
$$

2.3 Parameterized Complexity

The following section discusses a recent subfield of computational complexity theory referred to as parameterized complexity. This field originated from works of Downey and Fellows [20, 21] (see also [22, 27]).

2.3.1 Parameterized Reductions

A parameterized problem is a set of ordered pairs of the form (x, k) where k is referred to as the parameter. Any parameterized problem can be identified with 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. For short, we write $(k\text{-}\mathcal{A})$ and $(k\text{-}\mathcal{B})$ to denote the infinite families, respectively. We say that $(k\text{-}\mathcal{A})$ is **fpt-reducible** to $(k\text{-}\mathcal{B})$ if there exists an infinite family of reduction functions (r_k) and functions f and g such that for all $k \in \mathbb{N}$ and all instances x of $k\text{-}\mathcal{A}$, we have

$$x \in k\text{-}\mathcal{A} \iff r_k(x) \in f(k)\text{-}\mathcal{B}$$

and there exists a constant c such that for all $k \in \mathbb{N}$,

$$r_k \text{ is computable in } g(k) \cdot n^c \text{ time.}$$

We say that the fpt-reduction is **logspace** if there exists a function h such that

$$r_k \text{ is computable using } c \cdot \log(n) + h(k) \cdot o(\log(n)) \text{ bit cells of memory.}$$

Further, we say that the ftp-reduction is **uniform** if the infinite family of reduction functions is effectively computable. In parameterized complexity, what we refer to as a uniform ftp-reduction is the most standard and commonly used notion of reduction [27].

An **LBL-reduction** is a special kind of fpt-reduction such that for all $k \in \mathbb{N}$, $f(k) = k$. This means that the reduction exactly preserves the parameter[1]. 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 $O(n^d)$ where d is independent of k. All of the LBL-reductions that we encounter in this work can be improved to uniform LBL-reductions while many of them can be further improved to uniform logspace LBL-reductions.

We say that $(k\text{-}\mathcal{A})$ and $(k\text{-}\mathcal{B})$ are LBL-equivalent if $(k\text{-}\mathcal{A})$ is LBL-reducible to $(k\text{-}\mathcal{B})$ and $(k\text{-}\mathcal{B})$ is LBL-reducible to $(k\text{-}\mathcal{A})$. Notice that the concept of LBL-reduction forms a non-strict partial ordering on parameterized problems and the concept of LBL-equivalence forms an equivalence relation.

To simplify how one shows that an infinite family $(k\text{-}\mathcal{A})$ is LBL-reducible to $(D^T_{n^k})_{k\in\mathbb{N}}$, we have the following proposition.

Proposition 2.1. *Let an infinite family $(k\text{-}\mathcal{A})$ be given. If there exists c such that for every k, $k\text{-}\mathcal{A} \in \mathrm{DTIME}(n^{ck})$, then $(k\text{-}\mathcal{A})$ is LBL-reducible to $(D^T_{n^k})_{k\in\mathbb{N}}$.*

Sketch of proof. Let an infinite family $(k\text{-}\mathcal{A})$ be given. Suppose that there exists c such that for every k, $k\text{-}\mathcal{A} \in \mathrm{DTIME}(n^{ck})$. Let k be given. We describe how to reduce $k\text{-}\mathcal{A}$ to $D^T_{n^k}$.

[1]The notion of a parameter-preserving reduction from [40] is weaker only requiring that $f(k) = \mathsf{poly}(k)$. Further, the notion of a linear fpt-reduction from [14] is also weaker only requiring that $f(k) = O(k)$ and the modified version of linear fpt-reduction from [15] has a weaker requirement on the time complexity of the reduction.

Choose a deterministic Turing machine M that solves k-$\mathcal{A}$ in n^{ck} time. Therefore, M is n^{ck}-time bounded. We can modify M to get a machine M$'$ that is n^k-time bounded where M$'$ accepts a string w if and only if there is some natural number m such that w can be represented as $p \cdot x$ where p is a padding of length m^c and x is a string of length m that is accepted by M.

Now, let a string x of length m be given. The reduction takes x and maps to a pair consisting of a machine encoding M' and a string $p \cdot x$ where p is a padding of length m^c. We have that $x \in k$-$\mathcal{A}$ if and only if the pair is in $D^T_{n^k}$. Since M$'$ is fixed, this is a $O(n^c)$-time bounded reduction where c does not depend on k. □

The preceding proposition is for the complexity class P and acceptance problem $(D^T_{n^k})$. Similar results apply for all complexity classes and associated acceptance problems from subsection 2.2.3.

2.3.2 Parameterized Complexity Classes

Our primary focus is to relate the parameterized complexity of intersection non-emptiness problems to acceptance problems. To provide some perspective on where these problems fit within parameterzed complexity theory, we offer a brief overview of relevant complexity classes and refer the reader to [24, 27] for further details.

Consider a parameterized problem $(k$-$\mathcal{A})$. We say that $(k$-$\mathcal{A})$ is fixed parameter tractable if there exists a family of algorithms (a_k), a function g, and a constant c such that for all k,

$$a_k \text{ solves } k\text{-}\mathcal{A} \text{ in } g(k) \cdot n^c \text{ time.}$$

Further, we say that $(k$-$\mathcal{A})$ is uniform fixed parameter tractable and write $(k$-$\mathcal{A}) \in \mathrm{FPT}$ if the family of algorithms is effectively computable. Next, we consider characterizations of the first two levels of the W hierarchy. A parameterized problem $(k$-$\mathcal{A})$ is said to be $W[1]$-complete if it is uniform fpt-equivalent to the k-clique problem. Similarly, a parameterized problem $(k$-$\mathcal{A})$ is said to be $W[2]$-complete if it is uniform fpt-equivalent to the k-dominating set problem.

The acceptance problems from subsection 2.2.3 under uniform fpt-reductions[1] naturally define some additional parameterized complexity classes. In particular, we have the following relationships between acceptance problems and parameterized complexity classes[2].

$$(D^S_{k\log}) \ : \ \text{XL-complete}$$

$$(S^S_{k\log}) \ : \ \text{XSL-complete}$$

$$(N^S_{k\log}) \ : \ \text{XNL-complete}$$

$$(A^S_{k\log}) \ : \ \text{XAL-complete}$$

$$(Aux^S_{k\log}) \ : \ \text{XAuxL-complete}$$

$$(D^T_{n^k}) \ : \ \text{XP-complete}$$

$$(AltAux^S_{2^{n^k}}) \ : \ \text{XAltAuxL-complete}$$

$$(D^T_{2^{n^k}}) \ : \ \text{XEXP-complete}$$

Some of these parameterized problems are known to be LBL-equivalent. In particular, from [46], we have that L = SL. A careful analysis leads us to the following theorem.

Theorem 2.2. *$(D^S_{k\log})$ and $(S^S_{k\log})$ are LBL-equivalent.*

From [18, 11], we have that AL = AuxL = P. A careful analysis leads us to the following theorem.

Theorem 2.3. *$(A^S_{k\log})$, $(Aux^S_{k\log})$, and $(D^T_{n^k})$ are LBL-equivalent.*

From [43], we have that AltAuxL = EXP. A careful analysis leads us to the following theorem.

[1]For parameterized complexity classes XL, XSL, and XL, we restrict our attention to uniform logspace fpt-reductions.

[2]The closure of the acceptance problems on the left-hand side under fpt-reductions naturally characterizes their associated parameterized complexity classes.

Theorem 2.4. *($AltAux^S_{k\log}$) and ($D^T_{2^{n^k}}$) are LBL-equivalent.*

2.4 Intersection Non-Emptiness

2.4.1 General Formulation

Let $\mathcal{A}$ denote a class of automata. The intersection non-emptiness problem for $\mathcal{A}$, denoted by $\mathrm{IE}_{\mathcal{A}}$, consists of all finite lists of automata in $\mathcal{A}$ whose underlying languages have a non-empty intersection.

Formally, the input for $\mathrm{IE}_{\mathcal{A}}$ is an encoding of a finite list of automata. For each encoding, n will denote the length of the encoding and k will denote the number of automata in the list. By fixing the number of automata to k, one obtains intersection non-emptiness for k automata which we denote by k-$\mathrm{IE}_{\mathcal{A}}$.

2.4.2 Naming Conventions and Problems

We investigate the complexity of intersection non-emptiness problems for various classes of automata. To provide a general naming convention, we informally use subscripts to denote classes of automata.

Consider the following intersection non-emptiness problems and their corresponding classes of automata. We have the following problems for deterministic finite automata.

$\mathrm{IE}_{\mathcal{D}}$: deterministic finite automata

$\mathrm{IE}_{\mathcal{AC}}$: acyclic deterministic finite automata

$\mathrm{IE}_{\mathcal{TD}}$: tree shaped deterministic finite automata

We have the following problems for non-deterministic finite automata.

$\mathrm{IE}_{\mathcal{N}}$: non-deterministic finite automata

$\mathrm{IE}_{\mathcal{S}}$: symmetric finite automata

$\mathrm{IE}_{\mathcal{TN}}$: tree shaped non-deterministic finite automata

We have the following problem for non-string based automata.

$\mathrm{IE}_{\mathcal{T}}$: deterministic top-down tree automata

We have the following problems for mixed classes of automata.

$\mathrm{IE}_{1\mathcal{P}\mathcal{T}+\mathcal{T}}$: one pushdown tree automaton and tree automata

$\mathrm{IE}_{1\mathcal{P}+\mathcal{D}}$: one pushdown automaton and deterministic finite automata

$\mathrm{IE}_{1\mathcal{AC}+\mathcal{TD}}$: one acyclic deterministic finite automaton and tree shaped deterministic finite automata

3

Results for Space Complexity

3.1 Non-Deterministic Logspace

The following subsection is based on the author's ICALP 2014 paper [68].

3.1.1 Deterministic Finite Automata

Using the product construction, we can solve the intersection non-emptiness problem for k DFA's in $O(k \log(n))$ space.

Proposition 3.1. *There exists c such that for every k, k-IE$_{\mathcal{D}} \in$ NSPACE$^b(ck \log(n))$.*

Sketch of proof. We can solve k-IE$_{\mathcal{D}}$ by checking reachability in a product machine because there exists a path from an initial product state to a final product state if and only if there is a non-empty intersection. A state of the product machine can be stored as a string of $k \log(n)$ bits. Given the initial product state, we can non-deterministically guess which alphabet character comes next and transition to a new product state. We continue guessing until we've constructed a path from the initial product state to a final product state. Therefore, k-IE$_{\mathcal{D}}$ is solvable using at most $ck \log(n)$ bits for some constant c. □

By combining the preceding proposition and the technique of Proposition 2.1, we get the following corollary.

Corollary 3.2. *(k-IE$_{\mathcal{D}}$) is LBL-reducible to ($N^S_{k \log}$).*

We re-examine a construction from Karakostas, Lipton, and Viglas [36] to directly show that the acceptance problem for $k\log(n)$-space bounded non-deterministic binary Turing machines is reducible to the intersection non-emptiness problem for k DFA's (see also [39, 37]).

Theorem 3.3. *($N^S_{k\log}$) is LBL-reducible to (k-IE$_{\mathcal{D}}$).*

Proof. We describe a parameterized reduction from $N^S_{k\log}$ to k-IE$_{\mathcal{D}}$. Then, we discuss encoding details to justify it is a log-space parameterized reduction. Let a $k\log(n)$ space bounded non-deterministic Turing machine M of size n_{M} and an input string s of length n_s be given. Together, an encoding of M and s represent an arbitrary input for $N^S_{k\log}$. Let n denote the total size of M and s combined i.e. $n := n_{\mathrm{M}} + n_s$.

Our first task is to construct k DFA's, denoted by $\{D_i\}_{i\in[k]}$, each of size at most $p(n)$ for some fixed polynomial p such that M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(D_i)$ is non-empty. The DFA's will read in a string that represents a computation of M on s and verify that the computation is valid and accepting. The work tape of M will be split into k sections each consisting of $\log(n_s)$ sequential bits of memory. The ith DFA, D_i, will keep track of the ith section and verify that it is managed correctly. In addition, all of the DFA's will keep track of the input and work tape head positions. The following two concepts are essential to our construction.

A *section i configuration* of M is a tuple of the form

(state, input position, work position, ith section of work tape).

A *forgetful configuration* of M is a tuple of the form

(state, input position, work position, write bit).

The states of D_i are identified with section i configurations. Each D_i has a single initial state. We identify this initial state with the section i configuration of M that represents the initial input and work positions, a blank ith section of the work tape, and the initial state of M. We identify final states of D_i with section i configurations

of M that represent an accepting state of M. The alphabet characters are identified with forgetful configurations. Intuitively, the D_i's read in forgetful configurations that represent how the current bit cell should be manipulated and where to move the tape heads next.

The transitions for the DFA D_i are defined as follows. Let a forgetful configuration a and section i configurations r_1 and r_2 be given. It's possible that either the work tape position of r_1 is in the ith section, or the work tape position is in another section. In the first case, there is a transition from state r_1 with alphabet character a to state r_2 if the following are satisfied.

- Going from r_1 to r_2 represents a valid transition of M on input s.
- The ith section appropriately changes according to the write bit of a.
- We have that a and r_2 agree on state, input position, and work position.

In the second case, there is a transition from state r_1 with alphabet character a to state r_2 if the following are satisfied.

- We have r_1 and r_2 agree on the ith section of the work tape.
- We have a and r_2 agree on state, input position, and work position.

We assert without proof that for every string x, x represents a valid accepting computation of M on s if and only if $x \in \bigcap_{i\in[k]} \mathrm{L}(D_i)$. Therefore, M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(D_i)$ is non-empty.

We show that the D_i's have size at most $p(n)$ for some fixed polynomial p. Each D_i consists of a start state, a list of final states, and a list of transitions where each transition consists of two states and an alphabet character. Each state is represented by a section i configuration and each alphabet character is represented by a forgetful configuration. Therefore, in total there are $n_{\mathrm{M}} \cdot n_s \cdot k \log(n_s) \cdot 2^{\log(n_s)}$ section i configurations and $n_{\mathrm{M}} \cdot n_s \cdot k \log(n_s)$ forgetful configurations. Hence, there exists a fixed two variable polynomial q such that each D_i has size at most $q(n, k)$. Since k is fixed, one can blow up the degree of q to get a polynomial p such that p doesn't depend on k and each D_i has size at most $p(n)$.

It should be clear from the preceding that there is a fixed polynomial $t(n)$ such that for every k, $N^S_{k\log}$ is $t(n)$-time reducible to k-IE$_\mathcal{D}$. However, we want to show that there is a constant c such that for every k, $N^S_{k\log}$ is $c\log(n)$-space reducible to k-IE$_\mathcal{D}$. We accomplish this by describing how to print the string encoding of the D_i's to an auxiliary write only output tape using at most $c\log(n)$ space for some constant c.

We will describe how to print the transitions for each D_i and leave the remaining encoding details to the reader. We use a bit string i to represent the current DFA and two bit strings j_1 and j_2 to represent section i configurations. We iterate through every combination of i, j_1, and j_2. If D_i has a transition from j_1 to j_2, then we print (i, j_1, a, j_2) where a is the forgetful configuration such that j_2 extends a. We assert that checking whether to print (i, j_1, a, j_2) requires no more than $d\log(k) + d\log(n)$ bits for some constant d. Therefore, in printing the encoding of the D_i's, we use no more than $c\log(k) + c\log(n)$ bits for some constant c. For each k, when n is sufficiently large, the $\log(k)$ term goes away. It follows that for every k, $N^S_{k\log}$ is $c\log(n)$-space reducible to k-IE$_\mathcal{D}$. □

Corollary 3.4. *($N^S_{k\log}$) is LBL-equivalent to (k-IE$_\mathcal{D}$).*

Corollary 3.5. *(k-IE$_\mathcal{D}$) is* XNL*-complete.*

Corollary 3.6. *($N^S_{k\log}$) is LBL-equivalent to (k-IE$_\mathcal{N}$).*

Sketch of proof. Since DFA's are special cases of NFA's, we have that (k-IE$_\mathcal{D}$) is trivially LBL-reducible to (k-IE$_\mathcal{N}$). Also, we can apply the same approach from Proposition 3.1 to get a constant c such that for every k, k-IE$_\mathcal{N} \in$ NSPACE$^b(ck\log(n))$. Hence, (k-IE$_\mathcal{N}$) is LBL-reducible to ($N^S_{k\log}$).

Further, since we have that (k-IE$_\mathcal{D}$) and ($N^S_{k\log}$) are LBL-equivalent from Corollary 3.4, it follows that ($N^S_{k\log}$), (k-IE$_\mathcal{D}$), and (k-IE$_\mathcal{N}$) are all LBL-equivalent. □

3.1.2 Multi-Pass Automata

The following subsection is based on private communication between the author and Manuel Blum [70].

A k-pass automaton is a finite automaton that reads over the input string from left to right k times. At the end of each read, the automaton reads a special end character and then moves the tape head to the start of the input continuing the computation from the state where it left off from.

We consider the non-emptiness problem for k-pass deterministic finite automata. We denote this problem by k-PASS. There is a natural reduction from k-IE$_{\mathcal{D}}$ to k-PASS. We simply need to build a k-pass automaton such that each DFA is evaluated for one of the k passes.[1]

Proposition 3.7. (k-IE$_{\mathcal{D}}$) *is LBL-reducible to* (k-PASS).

Theorem 3.8. (k-PASS) *is LBL-reducible to* (k-IE$_{\mathcal{D}}$).

Proof. Let a k-pass automaton A be given. Let $|A|$ denote the number of states in A. We construct a finite list of finite automata $\{D_i\}_{i\in[k]}$ so that for each $i \in [k]$, $|D_i| = O(|A|^2)$.

Let $i \in [k]$ be given. The finite automaton D_i has roughly $|A|^2$ states. Aside from an initial segment of states, all of the states can be represented by a pair of states of A. That is, each state of D_i can be represented by a pair (s, t) where s is intended to represent the last state in the ith pass of A on an input string and t is intended to represent the current state in the ith pass of A on an input string. A state is final if $s = t$. However, in D_k, a state is only final if $s = t$ and s is a final state of A.

Essentially, the finite automata will first read in a bit string that encodes a list $\{s_i\}_{i\in[k]}$ of k states from A. For each $i \in [k]$, the state s_{i-1} represents the first state of the ith pass of A on an input[2] and s_i represents the last state reached by A of the ith pass of A on an input string. Next, the automata will read in the supposed input for A simulating what A would do. That is, for each $i \in [k]$, D_i simulates the ith pass of A on the supposed input verifying that the guessed last state s_i actually is the last state in the ith pass.

[1] Similar to the concatenation construction for building a DFA for the concatenation of two regular languages, we can connect the DFA state diagrams so that the final states of one DFA transition to the start state of the next DFA and so on.

[2] Except when $i = 1$ in which case the first state of the first pass is the start state of A.

We assert that there exists a string accepted by D_i for all $i \in [k]$ if and only if there exists a string accepted by A. To see this, one observes that any string accepted by each D_i corresponds with an evaluation of A on k-passes of an input string where the first and last states of each pass match up creating an accepting computation of A. Also, one observes that any accepting path through A corresponds with k-passes with associated first and last states where for each $i \in [k]$, the ith pass with corresponding states is accepted by the ith finite automaton. □

Corollary 3.9. $(N^S_{k\log})$ *is LBL-equivalent to* (k-PASS).

The non-emptiness problem for multi-pass automata is similar to the non-emptiness problem for two-way automata. The non-emptiness problem for two-way automata is a known PSPACE-complete problem [33, 29]. We assert without proof that $N^S_{k\log}$ is LBL-equivlaent to non-emptiness for k-turn two-way automata.

3.2 Deterministic Logspace

The following subsection is based on a collaboration between the author and Joseph Swernofsky to explore non-standard classes of finite automata.

3.2.1 Symmetric Automata

A symmetric finite automata (or SFA) is an NFA satisfying the following. Each alphabet character c has a corresponding alphabet character c^r such that if there is a transition from state p to q with alphabet character c, then there is a transition from state q to p with alphabet character c^r. We denote the intersection non-emptiness problem for SFA's by IE$_\text{S}$. Further, we denote the intersection non-emptiness problem for k SFA's by k-IE$_\text{S}$.

The concept of a symmetric Turing machine was introduced in [46]. We consider the complexity class SL consisting of languages accepted by symmetric logspace bounded Turing machines. By the equivalences from [57], we have that L = SL.

Proposition 3.10. *There exists c such that for every k, k-IE$_\text{S}$* $\in \text{DSPACE}^b(ck\log(n))$.

Sketch of proof. We solve k-IE$_{\mathbb{S}}$ by checking reachability in a product machine. We can do this because there exists a path from an initial product state to a final product state if and only if there is a non-empty intersection.

Notice that symmetric automata preserved their symmetric property under the product construction. That is, if there is a transition from a product state s_1 to a product state s_2 on an alphabet character a. There there is a reverse transition from s_2 to s_1 on the reverse alphabet character a^r because each coordinate of the product state is reversed on a^r by the defining property of symmetric automata.

Now, it was shown in [57] that we can solve reachability in an undirected graph in deterministic logarithmic space. In particular, it follows that there exists a constant c such that given an undirected graph with n vertices and two designated vertices v_1 and v_2, we can determine if there exists a path from v_1 to v_2 using at most $c\log(n)$ bits of memory.

Since the product machine has at most n^k product states, we just need to solve reachability in an reversible (undirected) graph with at most n^k vertices. We can do this using roughly $ck\log(n)$ bits of memory. Therefore, k-IE$_{\mathcal{D}}$ is solvable using at most $ck\log(n)$ bits for some constant c. □

Corollary 3.11. *(k-IE$_{\mathbb{S}}$) is LBL-reducible to ($D^S_{k\log}$).*

Theorem 3.12. *($S^S_{k\log}$) is LBL-reducible to (k-IE$_{\mathbb{S}}$).*

Proof. A symmetric Turing machine is a non-deterministic machine such that every transition has a corresponding reverse transition. We consider symmetric Turing machines with a read only input tape and a read/write binary work tape. There are three kinds of transitions based on the direction that the work tape head moves. There is a stationary transition that reads the current work tape cell and writes to it. There is a right moving transition that reads the current work tape cell and the cell to the right and writes to these two cells. Also, there is a left moving transition that reads the current work tape cell and the cell to the left and writes to these two cells. Similarly, the input head is able to peek one cell to the left or right based on the direction that it moves.

We describe a parameterized reduction from $S^S_{k\log}$ to k-IE$_\mathrm{S}$. We leave encoding details to the reader to justify that it is a log-space parameterized reduction. Let a $k\log(n)$ space bounded symmetric Turing machine M of size n_M and an input string s of length n_s be given. Together, an encoding of M and s represent an arbitrary input for $S^S_{k\log}$. Let n denote the total size of M and s combined i.e. $n := n_\mathrm{M} + n_s$.

Our first task is to construct k SFA's, denoted by $\{S_i\}_{i\in[k]}$, each of size at most $p(n)$ for some fixed polynomial p such that M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(S_i)$ is non-empty. The SFA's will read in a string that represents a computation of M on s and verify that the computation is valid and accepting. The work tape of M will be split into k sections each consisting of $\log(n_s)$ sequential bits of memory. The ith SFA, S_i, will keep track of the ith section and verify that it is managed correctly. In addition, all of the SFA's will keep track of the input and work tape head positions. The following two concepts are essential to our construction.

A *section i configuration* of M is a tuple of the form

(state, input position, work position, ith section of work tape).

An *informative configuration* of M is a tuple of the form

(source state, target state, input direction, work direction, read bits, write bits).

The states of S_i are identified with section i configurations. Each S_i has a single initial state. We identify this initial state with the section i configuration of M that represents the initial input and work positions, a blank ith section of the work tape, and the initial state of M. We identify final states of S_i with section i configurations of M that represent an accepting state of M. The alphabet characters are identified with informative configurations. Intuitively, the S_i's read in informative configurations that represent how the machine M could be manipulated in one step of the computation.

The transitions for the SFA S_i are defined as follows. Let an informative configuration a and section i configurations r_1 and r_2 be given. There is a transition from state r_1 with alphabet character a to state r_2 if the following are satisfied.

- There is a transition for M on input s corresponding to the informative configuration a.
- The tape positions move from r_1 to r_2 according to the tape directions of a.
- The source and target states of a match r_1 and r_2, respectively.
- When appropriate, the read and write bits of a match r_1 and r_2, respectively.
- The remaining bits from r_1 and r_2's section of the work tape match.

It remains to justify that these automata are SFA's. For each informative configuration, there is a corresponding reverse informative configuration. This reverse configuration is obtained by flipping the input and work directions, swapping the source and target states, and swapping the read bits and write bits. Since the machine M is symmetric, every transition has a corresponding reverse transition. By the construction of the finite automata, every transition has a corresponding reverse transition with the reverse alphabet character identified by the reverse informative configuration.

We assert without proof that for every string x, x represents a valid accepting computation of M on s if and only if $x \in \bigcap_{i\in[k]} \mathrm{L}(S_i)$. Therefore, M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(S_i)$ is non-empty.

We show that the S_i's have size at most $p(n)$ for some fixed polynomial p. Each S_i consists of a start state, a list of final states, and a list of transitions where each transition consists of two states and an alphabet character. Each state is represented by a section i configuration and each alphabet character is represented by a informative configuration. Therefore, in total there are $n_{\mathrm{M}} \cdot n_s \cdot k \log(n_s) \cdot 2^{\log(n_s)}$ section i configurations and $O(n_{\mathrm{M}}^2)$ informative configurations. Hence, there exists a fixed two variable polynomial q such that each S_i has size at most $q(n, k)$. Since k is fixed, one can blow up the degree of q to get a polynomial p such that p doesn't depend on k and each S_i has size at most $p(n)$. □

Corollary 3.13. *($D^S_{k\log}$) is LBL-equivalent to (k-IE$_{\mathrm{S}}$).*

Proof. From Theorem 2.2, we know that $(S^S_{k\log})$ and $(D^S_{k\log})$ are LBL-equivalent. By combining with Corollary 3.11 and Theorem 3.12, we obtain the desired result. □

Corollary 3.14. $(k\text{-IE}_S)$ *is* XL*-complete.*

4

Results for Time Complexity

The following section is based on the author's ICALP 2015 paper with Joseph Swernofsky [59].

4.1 Polynomial Time

4.1.1 Pushdown Automata

Using the product construction, we can solve the intersection non-emptiness problem for one pushdown automaton and k DFA's in $n^{O(k)}$ time.

Proposition 4.1. *There exists c such that for every k, k-$\mathrm{IE}_{1\mathcal{P}+\mathcal{D}} \in \mathrm{DTIME}(n^{ck})$.*

Proof. Non-emptiness for a single PDA is known to be solvable in polynomial time. Hence, we may choose c such that non-emptiness for PDA's is in $\mathrm{DTIME}(n^c)$.

Let k be given. Let an input consisting of one PDA and k DFA's be given. Let m denote the number of states from the largest automaton. Let n denote the total length of the input's string encoding. The product of the PDA and k DFA's is a PDA with at most m^{k+1} states and the product can be encoded by a string of length at most n^{k+1}. Now, we use the algorithm that decides non-emptiness for PDA's to solve non-emptiness for the product automaton in $O(n^{c(k+1)})$ time.

Since k is arbitrary, we have for all k, k-$\mathrm{IE}_{1\mathcal{P}+\mathcal{D}} \in \mathrm{DTIME}(n^{c(k+1)})$. We can choose a larger constant c' so that for all k, k-$\mathrm{IE}_{1\mathcal{P}+\mathcal{D}} \in \mathrm{DTIME}(n^{c'k})$. □

Corollary 4.2. *(k-$\mathrm{IE}_{1\mathcal{P}+\mathcal{D}}$) is LBL-reducible to ($D^T_{n^k}$).*

In the following theorem, we reduce acceptance for $k\log(n)$-space bounded auxiliary pushdown automata to intersection non-emptiness for one PDA and k DFA's. Our presentation is in the same format as that from subsection 3.1.1.

Theorem 4.3. *($Aux^{S}_{k\log}$) is LBL-reducible to (k-IE$_{1\mathcal{P}+\mathcal{D}}$).*

Proof. An auxiliary pushdown automaton has a stack, a two-way read-only input tape, and a single read/write work tape. We will restrict the read/write work tape to be binary and bound the amount of cells that the automaton can use in terms of the input length. In addition, we will only consider auxiliary pushdown automata where the stack alphabet is binary. Such restricted auxiliary PDA's are sufficient for carrying out the simulation in [18].

Let k be given. We will describe a reduction from $Aux^{S}_{k\log}$ to k-IE$_{1\mathcal{P}+\mathcal{D}}$. Let a $k\log(n)$-space bounded auxiliary pushdown automaton M of size n_{M} and an input string x of length n_x be given. Together, an encoding of M and x represent an arbitrary input for $Aux^{S}_{k\log}$. Let n denote the total size of M and x combined i.e. $n := n_{\mathrm{M}} + n_x$.

Our task is to construct one PDA and k DFA's, denoted by PD and $\{D_i\}_{i\in[k]}$, each of size at most $O(n^c)$ for some fixed constant c such that M accepts x if and only if $\mathrm{L}(PD) \cap \bigcap_{i\in[k]} \mathrm{L}(D_i)$ is non-empty. The automata will read in a string that represents a computation of M on x and verify that the computation is valid and accepting. The PDA PD will verify that the stack is managed correctly while the DFA's will verify that the work tape is managed correctly. In particular, the work tape of M will be split into k sections each consisting of $\log(n_x)$ sequential bits of memory. The ith DFA, D_i, will keep track of the ith section and verify that it is managed correctly. In addition, all of the DFA's will keep track of the tape head positions.

The following two concepts are essential to our construction.
A *section i configuration* of M is a tuple of the form:

(state, input position, work position, ith section of work tape).

A *forgetful configuration* of M is a tuple of the form:

(state, input position, work position, write bit, stack action, top bit).

The alphabet symbols are identified with forgetful configurations. The PDA PD only has two states. When it reads a forgetful configuration a, if a represents the top of the stack correctly, then PD loops in the initial/accepting state and pushes or pops based on the stack instruction that a represents. Otherwise, PD goes to the dead/rejecting state.

The states for the D_i's are identified with section i configurations. Each D_i has a single initial state. We identify this initial state with the section i configuration of M that represents the initial input and work positions, a blank ith section of the work tape, and the initial state of M. The final states of D_i represent accepting configurations of M.

Informally, the transitions are defined as follows. For each D_i, there is a transition from state r_1 to state r_2 with symbol a if a validly represents how the state and partial tapes for r_1 and r_2 could be manipulated in one step for the computation of M on input x. It's important to notice that in order to determine if there is a transition, the stack action and top bit of the stack must be taken into account.

We assert without proof that for every string y, y represents a valid accepting computation of M on x if and only if $y \in \mathrm{L}(PD) \cap \bigcap_{i\in[k]} \mathrm{L}(D_i)$. Therefore, M accepts x if and only if $\mathrm{L}(PD) \cap \bigcap_{i\in[k]} \mathrm{L}(D_i)$ is non-empty. By bounding the total number of section i configurations, one can show there exists a fixed two variable polynomial q such that each D_i has at most $q(n, k)$ states. Therefore, there is a constant d that does not depend on k such that each D_i has size at most $O(n^d)$ where k is treated as a constant. Further, we can compute each D_i's transition table by looping through every combination of a pair of states and an alphabet symbol and marking the valid combinations. The number of possible combinations is a fixed polynomial blow-up from n^d. Therefore, we can compute the transition tables in $O(n^c)$ time for some slightly larger constant c that does not depend on k.

Since k was arbitrary, we have that for every k, there is an $O(n^c)$-time reduction from $Aux^S_{k\log}$ to k-IE$_{1\mathcal{P}+\mathcal{D}}$. □

In the preceding reduction, it was surprising that the PDA had a fixed number of states. Even more surprisingly, one could convert the automata constructed in the reduction to automata with a binary input alphabet. In doing so, the PDA can be made fixed. In other words, there is a fixed deterministic pushdown automaton for which the intersection non-emptiness problem is hard.

By applying techniques from Cook [18] we obtain the following corollary.

Corollary 4.4. $(D^T_{n^k})$ *is LBL-equivalent to* (k-IE$_{1\mathcal{P}+\mathcal{D}}$).

Proof. From Theorem 2.3, we know that $(Aux^S_{k\log})$ and $(D^T_{n^k})$ are LBL-equivalent. By combining with Corollary 4.2 and Theorem 4.3, we obtain the desired result. □

Corollary 4.5. (k-IE$_{1\mathcal{P}+\mathcal{D}}$) *is* XP*-complete.*

4.1.2 Multi-Stack Pushdown Automata

A two-stack pushdown automaton can simulate a Turing machine. Therefore, the non-emptiness problem for such machines is undecidable. However, we can restrict how and when the machines can access their stacks to obtain classes of machines whose non-emptiness problems are decidable [49]. In particular, we discuss the k-phase switches restriction. This restriction forces a machine to designate a stack for popping. In other words, a restricted machine can push to any stack, but only pop from the designated stack. The k refers to how many times the machine can switch which stack is designated. We refer to a machine with such a restriction as a multi-stack pushdown automaton with k-phase switches. For background on such machines, we refer the reader to [62]. We also investigate what we refer to as the dual class of machines. These machines can pop from any stack, but can only push to the designated stack.

We denote the non-emptiness problem for multi-stack pushdown automata with k-phase switches by k-MPDA. Similarly, we denote the non-emptiness problem for the dual machines by k-*co*-MPDA.

Proposition 4.6. *There exists c such that for every k,* k-MPDA *and* k*-co-*MPDA $\in$ DTIME(n^{c2^k}).

Sketch of proof. In [49], it was shown that k-MPDA and k-*co*-MPDA $\in$ DTIME($n^{O(2^k)}$) by a reduction to non-emptiness for graph automata with bounded tree width and further to non-emptiness for tree automata. □

Corollary 4.7. (k-MPDA) *and* (k-*co-*MPDA) *are LBL-reducible to* ($D^T_{n^{2^k}}$).

Reductions have been introduced to show the hardness of related non-emptiness problems. In particular, a reduction was introduced to show that the non-emptiness problem for a related class of infinite automata has a double exponential time lower bound [42]. In addition, a reduction was introduced to show that the non-emptiness problem for ordered multi-stack pushdown automata has a double exponential time lower bound [3].

In the following theorem, we reduce intersection non-emptiness for one PDA and 2^k DFA's to non-emptiness for multi-stack pushdown automata with k-phase switches.

Theorem 4.8. (2^k-IE$_{1\mathcal{P}+\mathcal{D}}$) *is LBL-reducible to* (k-MPDA).

Sketch of proof. Let an input for 2^k-IE$_{1\mathcal{P}+\mathcal{D}}$ consisting of a PDA and 2^k DFA's be given. We will describe how to construct a multi-stack pushdown automaton M with k-phase switches whose language is non-empty if and only if the PDA and DFA's languages have a non-empty intersection.

The machine M will have k stacks. It will read its input and copy it onto all of the stacks besides the first stack. While it is reading the input, the first stack will be used to simulate the PDA on the input. Then, it will repeat the following procedure until each of the stacks have been designated once.

The procedure consists of popping from the designated stack and pushing what is being popped onto all of the other stacks. While it is popping, it is also simulating one DFA per copy of the input string or simulating one DFA in reverse per copy of the reversal of the input string. This will eventually create exponentially many

copies of the input string followed by the reversal of the input string and lead to simulating each DFA or reversal on one of the copies.

If the PDA and all of the DFA's accept, then M will accept. Otherwise, M will reject. In total, we are able to simulate one PDA and $O(2^k)$ DFA's using only k-phase switches. Also, the size of M will be approximately the sum of the sizes of the PDA and DFA's. □

A related reduction can be given for the dual machines.

Theorem 4.9. (2^k-IE$_{1\mathcal{P}+\mathcal{D}}$) *is LBL-reducible to* (k-*co*-MPDA)*.*

Sketch of proof. Let an input for 2^k-IE$_{1\mathcal{P}+\mathcal{D}}$ consisting of a PDA and 2^k DFA's be given. We will describe how to construct a dual machine with k-*co*-phase switches that accepts some input if and only if the PDA and DFA's languages have a non-empty intersection.

The MPDA will have k stacks. It will read an input consisting of 2^k separated strings and repeat the following procedure. As it is reading the input, it will copy the strings onto the designated stack. In addition, while reading, it will trade-off between simulating one DFA per string and simulating one DFA in reverse per string. Eventually, the MPDA will non-deterministically guess that it reached the midpoint i.e. the point when the designated stack's height is equal to the length of what still needs to be read on the input tape. When this happens, it will designate a new stack. Now, as it continues reading the input, it will pop the previously designated stacks and make sure that the strings on the input tape match the strings on these stacks while still repeating this procedure from the beginning on the newly designated stack.

Essentially, this procedure allows the MPDA to read in a sequence of exponentially many separated strings and use the stacks to verify that the strings are all equivalent modulo reversal. All the while, the MPDA is simulating one DFA per string or simulating one DFA in reverse per reversal of the string.

Finally, when the procedure is finished and the end of the input is reached, all the stacks are empty besides one stack with exactly one copy of the string. We can designate a new stack to simulate the PDA on the one remaining copy of the string.

It is alright for us to designate a new stack to simulate the PDA because the end of the input was reached and we already verified that all the strings on the input tape are the same modulo reversal.

If the PDA and all of the DFA's accept, then the MPDA will accept. Otherwise, it will reject. In total, we are able to simulate one PDA and $O(2^k)$ DFA's using only k-*co*-phase switches. Also, the size of the MPDA will be approximately the sum of the sizes of the PDA and DFA's. □

Corollary 4.10. (k-MPDA)*,* (k-*co*-MPDA)*, and* $(D^T_{n^{2^k}})$ *are LBL-equivalent.*

Proof. From Corollary 4.4, we know that $(2^k\text{-IE}_{1\mathcal{P}+\mathcal{D}})$ and $(D^T_{n^{2^k}})$ are LBL-equivalent. By combining with Corollary 4.7, Theorem 4.8, and Theorem 4.9, we obtain the desired result. □

Corollary 4.11. *Both* (k-MPDA) *and* (k-*co*-MPDA) *are* XP-*complete.*

4.1.3 Tree Automata

It is known that the general intersection non-emptiness problem for deterministic top-down tree automata is EXP-complete [17]. For background on decision problems for tree automata, we refer the reader to [17] and [50].

Proposition 4.12. *There exists* c *such that for every* k, $k\text{-IE}_{\mathcal{T}} \in \text{ASPACE}^b(ck\log(n))$.

Sketch of proof. We can solve $k\text{-IE}_{\mathcal{T}}$ by using alternation to check if there is a tree accepted by the product tree automaton. A state of the product machine can be stored as a string of $k\log(n)$ bits. Given such a state, we can non-deterministically guess which alphabet character comes next. Based on this character, we transition to a set of possible product states. Then, we apply a universal quantifier to continue the tree for each product state in this set independently. We repeat this process of alternation. Once an accepting product state is reached, we accept. Therefore, $k\text{-IE}_{\mathcal{T}}$ is solvable using at most $ck\log(n)$ bits for some constant c. □

Corollary 4.13. ($k\text{-IE}_{\mathcal{T}}$) *is LBL-reducible to* $(A^S_{k\log})$.

In the following theorem, we reduce acceptance for $k\log(n)$-space bounded alternating Turing machines to intersection non-emptiness for k tree automata. Related reductions can be found in [65] and briefly described in [17]. Our presentation is in the same format as that from subsection 3.1.1.

Theorem 4.14. $(A^S_{k\log})$ *is LBL-reducible to* $(k\text{-IE}_{\mathcal{T}})$.

Proof. An alternating Turing machine has existential states and universal states. Therefore, there are existential configurations and universal configurations. An existential configuration c leads to an accepting configuration if and only if there exists a valid transition out of c that leads to an accepting configuration. A universal configuration c leads to an accepting configuration if and only if every valid transition out of c leads to an accepting configuration. We will only consider alternating machines such that no universal configuration can have more than two valid outgoing transitions. We assert without proof that any alternating machine can be unraveled with intermediate universal states to satisfy this property in such a way that there is no more than a polynomial blow-up in the number of states.

Let k be given. We will describe a reduction from $A^S_{k\log}$ to $k\text{-IE}_{\mathcal{T}}$. Let a $k\log(n)$-space bounded alternating Turing machine M of size n_{M} and an input string x of length n_x be given. Together, an encoding of M and x represent an arbitrary input for $A^S_{k\log}$. Let n denote the total size of M and x combined i.e. $n := n_{\mathrm{M}} + n_x$.

Our task is to construct k top-down deterministic tree automata, denoted by $\{T_i\}_{i\in[k]}$, each of size at most $O(n^c)$ for some fixed constant c such that M accepts x if and only if $\bigcap_{i\in[k]} \mathrm{L}(T_i)$ is non-empty. The tree automata will read in a labeled tree that represents a computation of M on x and verify that the computation is valid and accepting. The work tape of M will be split into k sections each consisting of $\log(n_x)$ sequential bits of memory. The ith tree automaton, T_i, will keep track of the ith section and verify that it is managed correctly. In addition, all of the tree automata will keep track of the tape head positions.

The following two concepts are essential to our construction.

A *section i configuration* of M is a tuple of the form:

(state, input position, work position, ith section of work tape).

A *forgetful configuration* of M is a tuple of the form:

(state, input position, work position, write bit).

The states of T_i are identified with section i configurations. Each T_i has a single initial state. We identify this initial state with the section i configuration of M that represents the initial input and work positions, a blank ith section of the work tape, and the initial state of M. The alphabet consists of symbols of arity 0, 1, and 2 such that each arity 0 symbol represents an accepting forgetful configuration, each arity 1 symbol represents an arbitrary forgetful configuration, and each arity 2 symbol represents a pair of forgetful configurations. We won't need any symbols of arity larger than 2 because each universal configuration has at most two outgoing transitions.

We say that a section i configuration r extends a forgetful configuration a if r agrees with a on state, input position, and work position. We say that a section i configuration r_1 transitions to a section i configuration r_2 on input x if either the work position for r_1 is in the ith section and r_2 correctly represents how the tape positions and the ith section could change in one step of the computation on x or r_1 is not in the ith section and r_1 and r_2 agree on the ith section of the work tape.

For each T_i, we have the following transitions. Each arity 0 symbol a accepts on a state r if and only if r extends a and a represents an accepting state of M. Each arity 1 symbol a transitions from a state r_1 to a state r_2 if and only if (i) r_1 transitions to r_2 on input x (consistently with a's write bit), (ii) r_2 extends a, and (iii) if r_1 is a universal configuration and the work position of r_1 is in the ith section, then r_1 can only transition to r_2 on input x. Each arity 2 symbol (a_1, a_2) transitions from a state r to a pair of distinct states (r_1, r_2) if and only if r transitions to r_1 on input x, r transitions to r_2 on input x, r_1 extends a_1, and r_2 extends a_2.

We assert without proof that for every labeled tree y, y represents a valid accepting computation of M on x if and only if $y \in \bigcap_{i\in[k]} \mathrm{L}(T_i)$. Therefore, M accepts x if and only if $\bigcap_{i\in[k]} \mathrm{L}(T_i)$ is non-empty. By bounding the total number of section i configurations, one can show there exists a fixed two variable polynomial q such that each T_i has at most $q(n,k)$ states. Therefore, there is a constant d that does not depend on k such that each T_i has size at most $O(n^d)$ where k is treated as a constant. Further, we can compute each T_i's transition table by looping through every combination of a pair of states and an alphabet symbol and marking the valid combinations. The number of possible combinations is a fixed polynomial blow-up from n^d. Therefore, we can compute the transition tables in $O(n^c)$ time for some slightly larger constant c that does not depend on k.

Since k was arbitrary, we have that for every k, there is an $O(n^c)$-time reduction from $A^S_{k\log}$ to k-IE$_{\mathcal{T}}$. □

Theorem 4.15. *($D^T_{n^k}$) is LBL-equivalent to (k-IE$_{\mathcal{T}}$).*

Proof. From Theorem 2.3, we know that ($A^S_{k\log}$) and ($D^T_{n^k}$) are LBL-equivalent. By combining with Corollary 4.13 and Theorem 4.14, we obtain the desired result. □

Theorem 4.16. *(k-IE$_{\mathcal{T}}$) is* XP*-complete.*

4.2 Exponential Time

4.2.1 Pushdown Tree Automata

A pushdown tree automaton is essentially a tree automaton with auxiliary storage in the form of a stack. This notion was introduced in [30]. In general, a pushdown tree automaton has a special kind of stack that stores data in the form of a tree. However, we focus on what are referred to as restricted pushdown tree automata. Such an automaton has a stack in the usual sense. That is, the stack stores a string and behaves according to the standard push and pop operations.

The emptiness problem for pushdown tree automata has been considered by several authors. Following from [66, 41], there is a consensus that this problem is EXP-complete [2]. Also, from [53, 60] we have that the emptiness problem for context-free tree grammars is EXP-complete.

We consider the intersection non-emptiness problem for one pushdown tree automata and k tree automata. We denote this problem by k-IE$_{1\mathcal{PT}+\mathcal{T}}$. Rather than following the approaches from previous work on pushdown tree automata, we proceed by introducing a natural connection between pushdown tree automata and alternating auxiliary pushdown automata.

Proposition 4.17. *There exists c such that for every k,*

$$k\text{-IE}_{1\mathcal{PT}+\mathcal{T}} \in \text{AltAuxSPACE}^b(ck\log(n)).$$

Sketch of proof. We can solve k-IE$_{1\mathcal{PT}+\mathcal{T}}$ by using alternation to check if there is a tree accepted by the product pushdown tree automaton. A state of the product machine can be stored as a string of $k\log(n)$ bits. Given such a state, we can non-deterministically guess which alphabet character comes next. Based on this character, the product automaton would transition to a set of possible product states each with their own stack configuration. We apply a universal quantifier to continue the tree for each product state with the corresponding stack configuration. We repeat this process of alternation. Once an accepting product state is reached, we accept. Therefore, k-IE$_{1\mathcal{PT}+\mathcal{T}}$ is solvable using at most $ck\log(n)$ bits for some constant c. □

Corollary 4.18. *(k-IE$_{1\mathcal{PT}+\mathcal{T}}$) is LBL-reducible to ($AltAux^S_{k\log}$).*

Theorem 4.19. *($AltAux^S_{k\log}$) is LBL-reducible to (k-IE$_{1\mathcal{PT}+\mathcal{T}}$).*

Proof. An alternating auxiliary pushdown automaton is an alternating machine with a stack, a two-way read-only input tape, and a single read/write work tape. We will restrict the read/write work tape to be binary and bound the amount of cells that the automaton can use in terms of the input length. In addition, we will only consider alternating auxiliary pushdown automata where the stack

alphabet is binary and no universal configuration has more than two valid outgoing transitions. Such restricted alternating auxiliary PDA's are sufficient for carrying out the simulation in [43].

Let k be given. We will describe a reduction from $AltAux^{S}_{k\log}$ to k-$\mathrm{IE}_{1\mathcal{P}\mathcal{T}+\mathcal{T}}$. Let a $k\log(n)$-space bounded alternating Turing machine M of size n_{M} and an input string x of length n_x be given. Together, an encoding of M and x represent an arbitrary input for $AltAux^{S}_{k\log}$. Let n denote the total size of M and x combined i.e. $n := n_{\mathrm{M}} + n_x$.

Our task is to construct one top-down deterministic pushdown tree automaton and k top-down deterministic tree automata, denoted by PT and $\{T_i\}_{i\in[k]}$, each of size at most $O(n^c)$ for some fixed constant c such that M accepts x if and only if $\mathrm{L}(PT)\cap\bigcap_{i\in[k]}\mathrm{L}(T_i)$ is non-empty. The tree automata will read in a labeled tree that represents a computation of M on x and verify that the computation is valid and accepting. The pushdown tree automaton PT will verify that the stack is managed correctly down every branch of the computation while the tree automata will verify that the work tape is managed correctly down every branch of the computation. In particular, the work tape of M will be split into k sections each consisting of $\log(n_x)$ sequential bits of memory. The ith tree automaton, T_i, will keep track of the ith section and verify that it is managed correctly. In addition, all of the tree automata will keep track of the tape head positions.

The following two concepts are essential to our construction.
A *section i configuration* of M is a tuple of the form:

(state, input position, work position, ith section of work tape).

A *forgetful configuration* of M is a tuple of the form:

(state, input position, work position, write bit, stack action, top bit).

The alphabet consists of symbols of arity 0, 1, and 2 such that each arity 0 symbol represents an accepting forgetful configuration, each arity 1 symbol represents an arbitrary forgetful configuration, and each arity 2 symbol represents a pair of

forgetful configurations. We won't need any symbols of arity larger than 2 because each universal configuration has at most two outgoing transitions.

The pushdown tree automaton PT only has two states. When it reads an arity 0 symbol a, if a represents the top of the stack correctly, then PT accepts. When it reads an arity 1 symbol a, if a represents the top of the stack correctly, then PT loops in the initial/accepting state and pushes or pops based on the stack instruction that a represents. Otherwise, PT goes to the dead/rejecting state. When it reads an arity 2 symbol (a_1, a_2), if a_1 and a_2 represent the top of the stack correctly, then PT branches to a configuration corresponding to a_1 and a configuration corresponding to a_2 so that PT loops in the initial/accepting state and pushes or pops based on the stack instruction that a_1 or a_2 represents, respectively. Otherwise, PT goes to the dead/rejecting state for both branches.

The states of T_i are identified with section i configurations. Each T_i has a single initial state. We identify this initial state with the section i configuration of M that represents the initial input and work positions, a blank ith section of the work tape, and the initial state of M.

We say that a section i configuration r extends a forgetful configuration a if r agrees with a on state, input position, and work position. We say that a section i configuration r_1 transitions to a section i configuration r_2 on input x with stack bit b if either the work position for r_1 is in the ith section and r_2 correctly represents how the tape positions and the ith section could change in one step of the computation on x with top of stack b or r_1 is not in the ith section and r_1 and r_2 agree on the ith section of the work tape.

For each T_i, we have the following transitions. Each arity 0 symbol a accepts on a state r if and only if r extends a and a represents an accepting state of M. Each arity 1 symbol a transitions from a state r_1 to a state r_2 if and only if (i) r_1 transitions to r_2 on input x with a's top bit (consistently with a's write bit and stack action), (ii) r_2 extends a, and (iii) if r_1 is a universal configuration and the work position of r_1 is in the ith section, then r_1 can only transition to r_2 on input x with a's top bit. Each arity 2 symbol (a_1, a_2) transitions from a state r to a pair of

distinct states (r_1, r_2) if and only if r transitions to r_1 on input x with a_1's top bit, r transitions to r_2 on input x with a_2's top bit, r_1 extends a_1, and r_2 extends a_2.

We assert without proof that for every labeled tree y, y represents a valid accepting computation of M on x if and only if $y \in \mathrm{L}(PT) \cap \bigcap_{i\in[k]} \mathrm{L}(T_i)$. Therefore, M accepts x if and only if $\mathrm{L}(PT)\cap\bigcap_{i\in[k]} \mathrm{L}(T_i)$ is non-empty. By bounding the total number of section i configurations, one can show there exists a fixed two variable polynomial q such that each T_i has at most $q(n, k)$ states. Therefore, there is a constant d that does not depend on k such that each T_i has size at most $O(n^d)$ where k is treated as a constant. Further, we can compute each T_i's transition table by looping through every combination of a pair of states and an alphabet symbol and marking the valid combinations. The number of possible combinations is a fixed polynomial blow-up from n^d. Therefore, we can compute the transition tables in $O(n^c)$ time for some slightly larger constant c that does not depend on k.

Since k was arbitrary, we have that for every k, there is an $O(n^c)$-time reduction from $AltAux^S_{k\log}$ to k-$\mathrm{IE}_{1\mathcal{P}\mathcal{T}+\mathcal{T}}$. □

Corollary 4.20. $(D^T_{2^{n^k}})$ *is LBL-equivalent to* (k-$\mathrm{IE}_{1\mathcal{P}\mathcal{T}+\mathcal{T}}$).

Proof. From Theorem 2.4, we know that $(AltAux^S_{k\log})$ and $(D^T_{2^{n^k}})$ are LBL-equivalent. By combining with Corollary 4.18 and Theorem 4.19, we obtain the desired result. □

Corollary 4.21. (k-$\mathrm{IE}_{1\mathcal{P}\mathcal{T}+\mathcal{T}}$) *is* XEXP*-complete.*

5

Results for Time-Space Complexity

5.1 Deterministic Linear Time and Logspace

5.1.1 Acyclic Automata

An acyclic finite automaton[1] is a finite automaton whose state diagram forms a directed acyclic graph (ignoring the dead state). Since these automata contain no directed cycles or loops, they can only accept finite languages. In particular, an acyclic finite automaton with at most n states cannot accept any string of length larger than $n-1$.

We investigate the intersection non-emptiness problem for deterministic acyclic finite automata. We denote this problem by $\text{IE}_{\mathcal{AC}}$. In Rampersad and Shallit (2010), it was shown that $\text{IE}_{\mathcal{AC}}$ is NP-complete [56]. We strengthen their result by investigating the parameterized problem k-$\text{IE}_{\mathcal{AC}}$.

Consider the acceptance problem for non-deterministic n-time and $k\log(n)$ space bounded Turing machines. We denote this problem by $N^{(T,S)}_{(n,k\log)}$.

Proposition 5.1. *There exists c such that for every k, k-$\text{IE}_{\mathcal{AC}} \in \text{NTISP}^b(n, ck\log(n))$.*

[1] A deterministic acyclic finite automaton over a binary input alphabet is similar to the concept of a reduced ordered binary decision diagram [8] and the concept of an oblivious read-once binary decision diagram [38].

Sketch of proof. We can solve k-IE$_{\mathcal{AC}}$ by checking reachability in a product machine. A state of the product machine can be stored as a string of $k\log(n)$ bits. Given such a state, we can non-deterministically guess which state comes next. Since the automata can only accept strings of length at most $n-1$, there exists a path from an initial state to a final state if and only if there exists a path from an initial state to a final state of length at most $n-1$. Therefore, since the path we are guessing has length at most $n-1$, k-IE$_{\mathcal{AC}}$ is solvable in $O(n)$ time using at most $ck\log(n)$ bits for some constant c. □

Corollary 5.2. *(k-IE$_{\mathcal{AC}}$) is LBL-reducible to ($N^{(T,S)}_{(n,k\log)}$).*

Theorem 5.3. *($N^{(T,S)}_{(n,k\log)}$) is LBL-reducible to (k-IE$_{\mathcal{AC}}$).*

Proof. Let an n-time and $k\log(n)$-space bounded non-deterministic Turing machine M of size n_{M} and an input string s of length n_s be given. Together, an encoding of M and s represent an arbitrary input for $N^{(T,S)}_{(n,k\log)}$. Let n denote the total size of M and s combined i.e. $n := n_{\mathrm{M}} + n_s$.

We previously investigated a reduction from $N^{S}_{k\log}$ to k-IE$_{\mathcal{D}}$. We can apply this construction to get k DFA's, denoted by $\{D_i\}_{i\in[k]}$, each of size at most $p(n)$ for some fixed polynomial p such that M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(D_i)$ is non-empty. However, the DFA's that we constructed might not be acyclic. We provide an additional step to this construction to make the DFA's acyclic.

We construct k acyclic DFA's, denoted by $\{D'_i\}_{i\in[k]}$, each of size at most $(n+1)\cdot p(n)$ such that M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(D'_i)$ is non-empty. For each $i \in [k]$, the states of D'_i are represented by an ordered pair (s, j) where s is a state of D_i and j is a non-negative integer between 0 and n. The start state is $(s_0, 0)$ where s_0 is the start state of D_i and the final states are of the form (s_f, j) where s_f is a final state of D_i. Further, for each $j \in [n]$, there is a transition from state $(s_1, j-1)$ to (s_2, j) on character c if there is a transition from s_1 to s_2 on character c in D_i. Also, there is a transition from (s, n) to the dead state on every input character c.

Since the construction essentially just takes each D_i and limits the length of accepted strings. We have that a string $x \in \mathrm{L}(D'_i)$ if and only if $x \in \mathrm{L}(D_i)$ and

x has length at most n. Since the machine M is n-time bounded, we have that M accepts s if and only if M accepts s in at most n steps. Hence, M accepts s if and only if there exists a string of length at most n in $\bigcap_{i\in[k]} \mathrm{L}(D_i)$. Therefore, M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(D'_i)$ is non-empty. □

Corollary 5.4. $(N^{(T,S)}_{(n,k\log)})$ *is LBL-equivalent to* $(k\text{-IE}_{\mathcal{AC}})$.

5.2 Alternating Linear Time and Logspace

5.2.1 Acyclic Tree Automata

An acyclic tree automaton is a top-down tree automaton that contains no cycle branches. A cycle branch is a sequence of distinct states $\{s_i\}_{i\in[m]}$ such that (1) for each $i \in [m-1]$, there exists a transition from s_i to a tuple of states that contains s_{i+1} and (2) there exists a transition from s_m to a tuple of states that contains s_1.

Since acyclic tree automata contain no cycle branches, they can only accept finite tree languages. In particular, an acyclic tree automaton with at most n states cannot accept any tree of height larger than $n-1$.

We investigate the intersection non-emptiness problem for deterministic acyclic tree automata. We denote this problem by $\mathrm{IE}_{\mathcal{AC}\mathcal{T}}$. Further, we denote the intersection non-emptiness problem for k deterministic acyclic tree automata by $k\text{-IE}_{\mathcal{AC}\mathcal{T}}$.

Consider the acceptance problem for alternating n-time and $k\log(n)$ space bounded Turing machines. We denote this problem by $A^{(T,S)}_{(n,k\log)}$.

Proposition 5.5. *There exists* c *such that for every* k, $k\text{-IE}_{\mathcal{AC}\mathcal{T}} \in \mathrm{ATISP}^b(n, ck\log(n))$.

Sketch of proof. We can solve $k\text{-IE}_{\mathcal{AC}\mathcal{T}}$ by using alternation to check if there is a tree accepted by the product tree automaton of height at most n. A state of the product machine can be stored as a string of $k\log(n)$ bits. Given such a state, we can non-deterministically guess which tuple of states come next. Then, we apply a universal quantifier to continue the tree for each state in the tuple independently. Since the automata can only accept trees of height at most $n-1$, there exists an accepted tree if and only if there exists an accepted tree of height at most $n-1$.

Therefore, we only need to repeat this process of alternation at most $n-1$ times. Therefore, k-IE$_{\mathcal{ACT}}$ is solvable in $O(n)$ time using at most $ck\log(n)$ bits for some constant c. □

Corollary 5.6. *(k-IE$_{\mathcal{ACT}}$) is LBL-reducible to ($A^{(T,S)}_{(n,k\log)}$).*

Theorem 5.7. *($A^{(T,S)}_{(n,k\log)}$) is LBL-reducible to (k-IE$_{\mathcal{ACT}}$).*

Proof. Let an n-time and $k\log(n)$-space bounded alternating Turing machine M of size n_{M} and an input string s of length n_s be given. Together, an encoding of M and s represent an arbitrary input for $A^{(T,S)}_{(n,k\log)}$. Let n denote the total size of M and s combined i.e. $n := n_{\mathrm{M}} + n_s$.

We previously investigated a reduction from $A^S_{k\log}$ to k-IE$_{\mathcal{T}}$. We can apply this construction to get k tree automata, denoted by $\{T_i\}_{i\in[k]}$, each of size at most $p(n)$ for some fixed polynomial p such that M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(T_i)$ is non-empty. However, the tree automata that we constructed might not be acyclic. We provide an additional step to this construction to make the tree automata acyclic.

We construct k acyclic tree automata, denoted by $\{T'_i\}_{i\in[k]}$, each of size at most $(n+1)\cdot p(n)$ such that M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(T'_i)$ is non-empty. For each $i \in [k]$, the states of T'_i are represented by an ordered pair (s, j) where s is a state of T_i and j is a non-negative integer between 0 and n. The start state is $(s_0, 0)$ where s_0 is the start state of T_i. For each $j \in [n] \cup \{0\}$, a state (s, j) accepts on an arity 0 symbol a, if s accepts on a in T_i. For each $j \in [n]$, a state $(s_1, j-1)$ transitions to a state (s_2, j) on arity 1 symbol a if s_1 transitions to s_2 on a in T_i. Also, a state $(s_1, j-1)$ transitions to an ordered pair of states $((s_2, j), (s_3, j))$ on arity 2 symbol a if s_1 transitions to ordered pair (s_2, s_3) on a in T_i.

Since the construction essentially just takes each T_i and limits the height of accepted trees. We have that a string $x \in \mathrm{L}(T'_i)$ if and only if $x \in \mathrm{L}(T_i)$ and x has height at most n. Since the machine M is n-time bounded, we have that M accepts s if and only if M accepts s in at most n steps. Hence, M accepts s if and only if there exists a tree of height at most n in $\bigcap_{i\in[k]} \mathrm{L}(T_i)$. Therefore, M accepts s if and only if $\bigcap_{i\in[k]} \mathrm{L}(T'_i)$ is non-empty. □

Corollary 5.8. $(A^{(T,S)}_{(n,k\log)})$ *is LBL-equivalent to* $(k\text{-IE}_{\mathcal{ACT}})$.

6

Results for the W Hierarchy

6.1 Results for W[1]

The following subsection is based on the author's unpublished work [69].

6.1.1 Tree Shaped Automata

A finite automaton is said to be tree shaped if its state diagram takes the shape of a tree.[1] More formally, a finite automaton is said to be **tree shaped**[2] if its state diagram (without the dead state) forms a rooted tree satisfying the following.

(1) the root of the tree is the start state

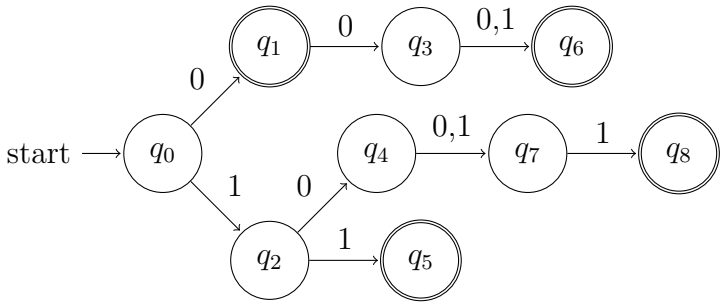
(2) all transitions are directed towards the leaves

(3) there are no loops

Consider the following example of a tree shaped DFA D_1:

[1] Tree shaped automata are not to be confused with tree automata. Tree shaped automata read in strings as input while tree automata read in labeled trees as input.

[2] A deterministic tree shaped automaton can be viewed as a reduced form of a decision tree where edges are allowed to be labelled with multiple alphabet characters. For background on binary decision trees, we refer the reader to [52].



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 automaton by its corresponding set of branches.

Let a natural number c be given. We denote the intersection non-emptiness problem for k tree shaped DFA's over the input alphabet $[c]$ by (c,k)-IE$_{\mathcal{TD}}$. Further, for every fixed c, we have a parameterized problem $((c,k)\text{-IE}_{\mathcal{TD}})_{k\in\mathbb{N}}$ which we abbreviate as $((c,k)\text{-IE}_{\mathcal{TD}})$. Although we do not pursue the topic of state complexity, there is an exponential blow-up for converting tree shaped NFA's to DFA's. As a result, we also consider the intersection non-emptiness problem for k tree shaped NFA's over the input alphabet $[c]$ which we denote by (c,k)-IE$_{\mathcal{TN}}$. In addition, consider the following classical parameterized problems.

- (c,k)-CLIQUE: Given a c-uniform hypergraph[1] H, does there exist a complete hypergraph with k vertices in H?
- (c,k)-WSAT: Given a c-CNF boolean formula ϕ, does there exist a satisfying assignment for ϕ with weight[2] exactly k?

We proceed by showing that these four parameterized problems are LBL-equivalent for all fixed c. First, by observing that every DFA is an NFA, we get the following.

[1] A c-uniform hypergraph is a hypergraph where each hyperedge is a set containing c vertices.
[2] The weight of an assignment is the same as the number of variables assigned the value 1.

Proposition 6.1. *For all fixed c, $((c,k)\text{-IE}_{\mathcal{TD}})$ is LBL-reducible to $((c,k)\text{-IE}_{\mathcal{TN}})$ with $O(n)$ instance blow-up.*

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

Theorem 6.2. *For all fixed c, $((c,k)\text{-IE}_{\mathcal{TN}})$ is LBL-reducible to $((c,k)\text{-CLIQUE})$ 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 uniform hypergraph clique problem to the weighted satisfiability problem.

Theorem 6.3. *For all fixed c, $((c,k)$-CLIQUE$)$ is LBL-reducible to $((c,k)$-WSAT$)$ 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 the weighted satisfiability problem to intersection non-emptiness for tree shaped DFA's.

Theorem 6.4. *For all fixed c, $((c,k)$-WSAT$)$ is LBL-reducible to $((c,k)$-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 6.5. *For all fixed c, the following are LBL-equivalent with $O(n^{3c})$ instance blow-up:* $((c,k)\text{-IE}_{\mathfrak{TD}})$, $((c,k)\text{-IE}_{\mathfrak{TN}})$, $((c,k)\text{-CLIQUE})$, *and* $((c,k)\text{-WSAT})$.

Corollary 6.6. *For all fixed c, $((c,k)\text{-IE}_{\mathfrak{TD}})$ is $W[1]$-complete.*

6.2 Results for W[NL]

6.2.1 Acyclic Automata

We investigate the intersection non-emptiness problem for one acyclic DFA and k tree shaped DFA's over a binary input alphabet. We denote this problem by $k\text{-IE}_{1\mathcal{AC}+\mathfrak{TD}}$.

Consider the parameterzied problems that are verifiable in linear time with read-only access to a $k\log(n)$ bit certificate. The closure of such problems under fpt-reductions defines the parameterized complexity class $W[\mathrm{P}]$. Similarly, one could consider space bounded verifiers. In particular, we consider the parameterzied problems that are non-deterministically verifiable using $\log(n)$ bits of memory with read-only access to a $k\log(n)$ bit certificate. The closure of such problems under fpt-reductions defines the parameterized complexity class $W[\mathrm{NL}]$.

Consider the acceptance problem for languages non-deterministically verifiable using $\log(n)$ bits of memory with read-only access to a $k\log(n)$ bit certificate. We denote this problem by $N^{(S,G)}_{(\log,k\log)}$ and denote the corresponding complexity measure by NSPGU^b where G stands for guesses.

Proposition 6.7. *There exists c such that for every k,*

$$k\text{-IE}_{1\mathcal{AC}+\mathcal{TD}} \in \mathrm{NSPGU}^b(c\log(n), k\log(n)).$$

Sketch of proof. Using only a $k\log(n)$ bit certificate, we can guess a branch from each of the k tree shaped DFA's. Then, using non-deterministic logspace, we check whether there exists an input string that matches each of the k branches and the acyclic DFA. We do this as follows.

We store the current state in the acyclic DFA and a counter with value at most n. To do this, we use roughly $2\log(n)$ bits of memory. Given such a state and count, we non-deterministically guess which alphabet character comes next. We go through each of the k branches and check to make sure that the alphabet character is value for the position with the current count. If it is, then we increment the counter and transition to the next state. We repeat this process until the count reaches n or the end of the branches is reached. If we are in a final state, then we accept.

Since the DFA's have a non-empty intersection if and only if there is a choice of branches that have a non-empty intersection with the acyclic DFA, $k\text{-IE}_{1\mathcal{AC}+\mathcal{TD}}$ is verifiable with a $k\log(n)$ bit certificate using at most $c\log(n)$ bits for some constant c. □

Corollary 6.8. *(k-IE$_{1\mathcal{AC}+\mathcal{TD}}$) is LBL-reducible to ($N^{(S,G)}_{(\log,k\log)}$).*

Theorem 6.9. *($N^{(S,G)}_{(\log,k\log)}$) is LBL-reducible to (k-IE$_{1\mathcal{AC}+\mathcal{TD}}$).*

Proof. Let a non-deterministic Turing machine verifier M using at most $\log(n)$ bits of memory and an input string x be given. We construct k tree shaped DFA's and an acyclic DFA such that the DFA's have a non-empty intersection if and only if there exists a satisfying certificate of $k\log(n)$ bits.

The idea is that the DFA's will read in the certificate and the computation. As the DFA's are reading the input, they collectively verify that it is valid and accepting. The k tree shaped DFA's will each store a portion of the certificate of length $\log(n)$. Then, the acyclic DFA will do the bulk of the work in verifying the computation.

To make this work, we need to interleave copies of the certificate between each configuration of the computation. In particular, the input string will have n configurations and between each pair of configurations it will have a copy of the $k\log(n)$ bit certificate. The tree shaped DFA's each verify that their portion of the certificate is consist from copy to copy while ignoring the configurations. Then, the acyclic DFA keeps track of the tape heads, the $\log(n)$ bits on the work tape, and the current state. Based on this information, the acyclic DFA can verify that the configurations appropriately transition from one to the next making a valid computation. It also verifies that the last configuration is accepting.

Together, the DFA's verify that an input string corresponds with a choice of certificate and valid and accepting computation. Therefore, the DFA's have a non-empty intersection if and only if such a certificate and computation exists. □

Corollary 6.10. *(k-IE$_{1\mathcal{AC}+\mathcal{TD}}$) is W[NL]-complete.*

7

Lower Bounds

The following chapter is loosely based on a combination of results from the author's works [68, 59, 69].

7.1 Unconditional Lower Bounds

7.1.1 Space Complexity

In the previous chapters, we focused on showing how solving intersection non-emptiness problems is equivalent to simulating Turing machine computations. In this section, we combine this equivalence with classical hierarchy theorems[1] to prove unconditional lower bounds.

The space hierarchy theorem is a classical result stating that for every space constructible function $s(n)$, $\mathrm{DSPACE}(o(s(n))) \subsetneq \mathrm{DSPACE}(s(n))$. In terms of binary Turing machines, this result can be improved to

$$\mathrm{DSPACE}^b((1-\varepsilon)\cdot s(n)) \subsetneq \mathrm{DSPACE}^b(s(n)).$$

For non-deterministic space, it gets a little more complicated. One applies the classical result that NL = *co*-NL to binary Turing machines to get that there exists c such that for every k, $\mathrm{NSPACE}^b(k\log(n)) \subseteq co\text{-}\mathrm{NSPACE}^b(ck\log(n))$. Further by

[1] The proof of the space and time hierarchy theorems are based on combining universal simulation and diagonalization.

applying the approach of universal simulation and diagonalization from the space hierarchy theorem, we get that there exists c such that for every k,

$$\text{NSPACE}^b(c \cdot k \cdot s(n))) \subsetneq \text{NSPACE}^b(k \cdot s(n)).$$

Paralleling this approach to the non-deterministic space hierarchy theorem for binary Turing machines, we have the following theorem.

Theorem 7.1. *There exist c_1 and c_2 such that for every k, $N^S_{k\log} \in \text{NSPACE}^b(c_1 k \log(n))$ and $N^S_{k\log} \notin \text{NSPACE}^b(c_2 k \log(n))$.*

Theorem 7.2. *If an infinite family (X_k) is logspace LBL-equivalent to $(N^S_{k\log})$, then there exist c_1 and c_2 such that for every k, $X_k \in \text{NSPACE}^b(c_1 k \log(n))$ and $X_k \notin \text{NSPACE}^b(c_2 k \log(n))$.*

Proof. Let an infinite family (X_k) be given. Suppose that (X_k) is logspace LBL-equivalent to $(N^S_{k\log})$ with reduction constants c_1 and c_2. Let k be given. Consider the $c_1 \log(n)$-space bounded reduction from X_k to $N^S_{k\log}$. By universal simulation, there is a non-deterministic binary Turing machine M that solves $N^S_{k\log}$ in $d_1 k \log(n)$ space for a constant d_1 that does not depend on k. If we combine the reduction with the machine M, we get a machine that solves X_k in $c_1 d_1 k \log(n)$ space. Therefore, $X_k \in \text{NSPACE}^b(c_1 d_1 k \log(n))$.

Consider the $c_2 \log(n)$-space bounded reduction from $N^S_{k\log}$ to X_k. By diagonalization, we get a constant d_2 that does not depend on k such that $N^S_{k\log} \notin \text{NSPACE}^b(d_2 k \log(n))$. Hence, we can't solve X_k in $\frac{d_2 k}{c_2} \log(n)$ space or else we would be able to combine such a solver with the reduction to show that $N^S_{k\log} \in \text{NSPACE}^b(d_2 k \log(n))$ which can't happen. Therefore, $X_k \notin \text{NSPACE}^b(\frac{d_2 k}{c_2} \log(n))$. □

By combining the LBL equivalences[1] from Chapter 3 with Theorem 7.2, we get the following near tight complexity bounds.[2]

[1] It is important to notice that all of the LBL-equivalences from Chapter 3 can be improved to logspace LBL-equivalences. See the proof of Theorem 3.3 for further details.

[2] Kasai and Iwata (1985) showed related lower bounds for non-deterministic logspace including Corollary 7.3 [37].

Corollary 7.3. *There exist c_1 and c_2 such that for every k, k-IE$_{\mathcal{D}} \in$ NSPACE$^b(c_1 k \log(n))$ and k-IE$_{\mathcal{D}} \notin$ NSPACE$^b(c_2 k \log(n))$.*

Corollary 7.4. *There exist c_1 and c_2 such that for every k, k-PASS $\in$ NSPACE$^b(c_1 k \log(n))$ and k-PASS $\notin$ NSPACE$^b(c_2 k \log(n))$.*

7.1.2 Time Complexity

The time hierarchy theorem is a classical result stating that for every time constructible function $t(n)$, DTIME$(o(\frac{t(n)}{\log(t(n))})) \subsetneq$ DTIME$(t(n))$. Paralleling this classical result, we have the following theorem.

Theorem 7.5. *There exist c_1 and c_2 such that for every k, $D^T_{n^k} \in$ DTIME$(n^{c_1 k})$ and $D^T_{n^k} \notin$ DTIME$(n^{c_2 k})$.*

Theorem 7.6. *If an infinite family (X_k) is LBL-equivalent to $(D^T_{n^k})$, then there exist c_1 and c_2 such that for every k, $X_k \in$ DTIME$(n^{c_1 k})$ and $X_k \notin$ DTIME$(n^{c_2 k})$.*

Proof. Let an infinite family (X_k) be given. Suppose that (X_k) is LBL-equivalent to $(D^T_{n^k})$ with reduction constants c_1 and c_2. Let k be given. Consider the $O(n^{c_1})$-time bounded reduction from X_k to $D^T_{n^k}$. By universal simulation, there is a deterministic Turing machine M that solves $D^T_{n^k}$ in $n^{d_1 k}$ time for a constant d_1 that does not depend on k. If we combine the reduction with the machine M, we get a machine that solves X_k in $O(n^{c_1 d_1 k})$ time. Therefore, $X_k \in$ DTIME$(n^{c_1 d_1 k})$.

Consider the $O(n^{c_2})$-time bounded reduction from $D^T_{n^k}$ to X_k. By diagonalization, we get a constant d_2 that does not depend on k such that $D^T_{n^k} \notin$ DTIME$(n^{d_2 k})$. Hence, we can't solve X_k in $O(n^{\frac{d_2 k}{c_2}})$ time or else we would be able to combine such a solver with the reduction to show that $D^T_{n^k} \in$ DTIME$(n^{d_2 k})$ which can't happen. Therefore, $X_k \notin$ DTIME$(n^{\frac{d_2 k}{c_2}})$. □

By combining the LBL equivalences from Chapter 4 with Theorem 7.6, we get the following near tight complexity bounds.

Corollary 7.7. *There exist c_1 and c_2 such that for every k, k-IE$_{1\mathcal{P}+\mathcal{D}} \in$ DTIME$(n^{c_1 k})$ and k-IE$_{1\mathcal{P}+\mathcal{D}} \notin$ DTIME$(n^{c_2 k})$.*

Corollary 7.8. *There exist c_1 and c_2 such that for every k, k-IE$_{\mathcal{T}} \in$ DTIME$(n^{c_1 k})$ and k-IE$_{\mathcal{T}} \notin$ DTIME$(n^{c_2 k})$.*

Corollary 7.9. *There exist c_1 and c_2 such that for every k:*

i) k-MPDA *and* k-*co*-MPDA $\in$ DTIME$(n^{c_1 2^k})$
ii) k-MPDA *and* k-*co*-MPDA $\notin$ DTIME$(n^{c_2 2^k})$.

7.2 Conditional Lower Bounds

7.2.1 Complexity Class Separations

In the following subsection we show the relationship between the collapse of classical complexity classes and the LBL-equivalence of acceptance problems. As corollaries, we get that the existence of time or space efficient algorithms for intersection non-emptiness problems implies complexity class separations.

The following shows that in order for NL and P to be equal, there must be a level-by-level equivalence between their corresponding acceptance problems.

Theorem 7.10. NL = P *if and only if* $(N^S_{k\log})$ *and* $(D^T_{n^k})$ *are logspace LBL-equivalent.*

Proof. Suppose that $(N^S_{k\log})$ and $(D^T_{n^k})$ are logspace LBL-equivalent. Since NL $\subseteq$ P, we just need to show that P $\subseteq$ NL. Let a language $X \in$ P be given. There exists some k such that $X \in$ DTIME(n^k). Consider the problem $D^T_{n^k}$. Since it is logspace reducible to $N^S_{k\log}$ and $N^S_{k\log} \in$ NL, we have that $D^T_{n^k} \in$ NL. By a trivial reduction from X to $D^T_{n^k}$, we have that $X \in$ NL.

For the other direction, suppose that NL = P. Since any $k\log(n)$-space bounded binary Turing machine can be simulated in roughly deterministic n^k time, we can perform a universal simulation so that $N^S_{k\log} \in$ DTIME(n^{ck}) for some constant c that does not depend on k. Therefore, $(N^S_{k\log})$ is logspace LBL-reducible to $(D^T_{n^k})$. Next, since $D^T_{n^2} \in$ P and NL = P, $D^T_{n^2} \in$ NL. Hence, there exists c such that $D^T_{n^2} \in$ NSPACE$^b(c\log(n))$.

Let k be given. We provide a logspace reduction from $D^T_{n^k}$ to $D^T_{n^2}$. Let an input for $D^T_{n^k}$ consisting of an n^k-time bounded machine M and an input x be given. We can modify M to get a n^2-time bounded[1] machine M′ and modify x to get a padded string $y\#x$ where the padding y has length at least $|x|^k$. Then, M′'s computation is as follows. It first breaks an input string into a padding of length m_1 and right part of length m_2. It verifies that $m_1 > m_2^k$. Then, it simulates M on the right part and accepts according to M's computation. Therefore, M accepts x if and only if M′ accepts $y\#x$.

By combining the reduction and the logspace algorithm for $D^T_{n^2}$ we have $D^T_{n^k} \in \mathrm{NSPACE}^b(c \cdot d \cdot k\log(n))$ for some overhead constant d that does not depend on k. Since k was arbitrary, we have that $(D^T_{n^k})$ is LBL-reducible to $(N^S_{k\log})$. □

Corollary 7.11. *If for every $a > 0$, there exists k such that $N^S_{k\log} \in \mathrm{DTIME}(n^{ak})$, then* NL ≠ P.

Proof. Suppose that for every $a > 0$, there exists k such that $N^S_{k\log} \in \mathrm{DTIME}(n^{ak})$. In addition, suppose for sake of contradiction that NL = P. By Theorem 7.10, $(N^S_{k\log})$ and $(D^T_{n^k})$ are logspace LBL-equivalent. Hence, there is a constant c such that for every k, there is a $c\log(n)$-space bounded reduction from $D^T_{n^k}$ to $N^S_{k\log}$. Therefore, this reduction is n^c-time bounded as well.

Next, if we choose $a = \frac{1-\varepsilon}{c}$, then there exists some k sufficiently large such that

$$N^S_{k\log} \in \mathrm{DTIME}(n^{\frac{(1-\varepsilon)\cdot k}{c}}).$$

By combining this algorithm with the preceding reduction, we get that

$$D^T_{n^k} \in \mathrm{DTIME}(n^{(1-\varepsilon)\cdot k}).$$

Therefore,

$$\mathrm{DTIME}(n^k) \subseteq \mathrm{DTIME}(n^{(1-\varepsilon)\cdot k})$$

which contradicts the time hierarchy theorem. □

By applying LBL-equivalences from Chapter 3, we get the following corollaries.

[1] It's worth noticing that this machine can be made $c \cdot n$-time bounded for some constant c.

Corollary 7.12. *If for every $a > 0$, there exists k such that k-$\mathrm{IE}_{\mathcal{D}} \in \mathrm{DTIME}(n^{ak})$, then* $\mathrm{NL} \neq \mathrm{P}$.

Corollary 7.13. *If for every $a > 0$, there exists k such that k-PASS $\in \mathrm{DTIME}(n^{ak})$, then* $\mathrm{NL} \neq \mathrm{P}$.

The following shows that in order for P and PSPACE to be equal, there must be a level-by-level equivalence between their corresponding acceptance problems.

Theorem 7.14. $\mathrm{P} = \mathrm{PSPACE}$ *if and only if $(D^T_{n^k})$ and $(D^S_{n^k})$ are LBL-equivalent.*

Proof. Suppose that $(D^T_{n^k})$ and $(D^S_{n^k})$ are logspace LBL-equivalent. Since $\mathrm{P} \subseteq \mathrm{PSPACE}$, we just need to show that $\mathrm{PSPACE} \subseteq \mathrm{P}$. Let a language $X \in \mathrm{PSPACE}$ be given. There exists some k such that $X \in \mathrm{DSPACE}(n^k)$. Consider the problem $D^S_{n^k}$. Since it is polynomial time reducible to $D^T_{n^k}$ and $D^T_{n^k} \in \mathrm{P}$, we have that $D^S_{n^k} \in \mathrm{P}$. By a trivial reduction from X to $D^S_{n^k}$, we have that $X \in \mathrm{P}$.

For the other direction, suppose that $\mathrm{P} = \mathrm{PSPACE}$. Since any deterministic n^k-time bounded Turing machine uses at most n^k space, we can perform a universal simulation so that $D^T_{n^k} \in \mathrm{DSPACE}(n^{ck})$ for some constant c that does not depend on k. Therefore, $(D^T_{n^k})$ is LBL-reducible to $(D^S_{n^k})$. Next, since $D^S_{n^2} \in \mathrm{PSPACE}$ and $\mathrm{P} = \mathrm{PSPACE}$, $D^S_{n^2} \in \mathrm{P}$. Hence, there exists c such that $D^T_{n^2} \in \mathrm{DTIME}(n^c)$.

Let k be given. We provide a polynomial time reduction from $D^S_{n^k}$ to $D^S_{n^2}$. Let an input for $D^S_{n^k}$ consisting of an n^k-space bounded machine M and an input x be given. We can modify M to get a n^2-space bounded[1] machine M$'$ and modify x to get a padded string $y\#x$ where the padding y has length at least $|x|^k$. Then, M$'$'s computation is as follows. It first breaks an input string into a padding of length m_1 and right part of length m_2. It verifies that $m_1 > m_2^k$. Then, it simulates M on the right part and accepts according to M's computation. Therefore, M accepts x if and only if M$'$ accepts $y\#x$.

By combining the reduction and the polynomial time algorithm for $D^T_{n^2}$ we have $D^T_{n^k} \in \mathrm{DTIME}(n^{c \cdot d \cdot k})$ for some overhead constant d that does not depend on k. Since k was arbitrary, we have that $(D^S_{n^k})$ is LBL-reducible to $(D^T_{n^k})$. □

[1] It's worth noticing that this machine can be made $(1 + o(1)) \cdot n$-space bounded.

Corollary 7.15. *If for every $a > 0$, there exists k such that $D^T_{n^k} \in \mathrm{DSPACE}(n^{ak})$, then* $\mathrm{P} \neq \mathrm{PSPACE}$.

Proof. Suppose that for every $a > 0$, there exists k such that $D^T_{n^k} \in \mathrm{DSPACE}(n^{ak})$. In addition, suppose for sake of contradiction that $\mathrm{P} = \mathrm{PSPACE}$. By Theorem 7.14, $(D^T_{n^k})$ and $(D^S_{n^k})$ are LBL-equivalent. Hence, there is a constant c such that for every k, there is a n^c-time bounded reduction from $D^S_{n^k}$ to $D^T_{n^k}$.

Next, if we choose $a = \frac{1-\varepsilon}{c}$, then there exists some k sufficiently large such that

$$D^T_{n^k} \in \mathrm{DSPACE}(n^{\frac{(1-\varepsilon)\cdot k}{c}}).$$

By combining this algorithm with the preceding reduction, we get that

$$D^S_{n^k} \in \mathrm{DSPACE}(n^{(1-\varepsilon)\cdot k}).$$

Therefore,

$$\mathrm{DSPACE}(n^k) \subseteq \mathrm{DSPACE}(n^{(1-\varepsilon)\cdot k})$$

which contradicts the space hierarchy theorem. □

By applying LBL-equivalences from Chapter 4, we get the following corollaries.

Corollary 7.16. *If for every $a > 0$, there exists k such that k-$\mathrm{IE}_{1\mathcal{P}+\mathcal{D}} \in \mathrm{DSPACE}(n^{ak})$, then* $\mathrm{P} \neq \mathrm{PSPACE}$.

Corollary 7.17. *If for every $a > 0$, there exists k such that k-$\mathrm{IE}_{\mathcal{T}} \in \mathrm{DSPACE}(n^{ak})$, then* $\mathrm{P} \neq \mathrm{PSPACE}$.

7.2.2 QBF-Hardness

The following subsection is adapted from a response by the author on a post from the cstheory stackexchange community titled "Deciding emptiness of intersection of regular languages in subquadratic time".

Theorem 7.18. *Let a natural number $k \geq 2$ be given. If k-$\mathrm{IE}_{\mathcal{D}} \in \mathrm{DTIME}(n^{\delta})$, then* $\mathrm{NSPACE}^b(n) \subseteq \mathrm{DTIME}(\mathsf{poly}(n) \cdot 2^{\frac{\delta n}{k}})$.

Proof. Suppose that we can solve intersection non-emptiness for k DFA's in $O(n^\delta)$ time. Let a non-deterministic n-space bounded binary Turing machine M be given. Let an input string x of length n be given. A computation of M on input x can be represented by a finite list of configurations. Each configuration consists of a state, a position on the input tape, a position on the work tape, and up to n bits of memory that represent the work tape.

Consider that the work tape is split into k block. In other words, we have k blocks each with $\frac{n}{k}$ work tape cells. Each configuration can be broken up into k pieces. For each $i \in [k]$, the ith piece consists of the state, the position on the input tape, the position on the work tape, and the $\frac{n}{k}$ bits from the ith block.

Next, for each i, we build a DFA D_i whose states are ith pieces of configurations. The alphabet characters are instructions that say which state to go to, how the tape heads should move, and how the work tape's active cell should be manipulated. The idea is that the DFA's read in a list of instructions corresponding to a computation of M on input x and together verify that it is valid and accepting. The DFA's always agree with each other on where the tape heads are because that information is stored in their states and the instruction of where it moves next is included in the input characters. Therefore, for each $i \in [k]$, we can have D_i verify that the instruction is appropriate when the work tape position is in the ith piece.

In total, there are at most $poly(n) \cdot 2^{\frac{n}{k}}$ states for each DFA and at most $poly(n)$ distinct alphabet characters. By the initial assumption, it follows that we can solve intersection non-emptiness for the k DFA's in $poly(n) \cdot 2^{\frac{\delta n}{k}}$ time. □

Determining whether a Quantified Boolean Formula evaluates to true is a classic PSPACE-complete problem. This problem is a natural extension of boolean satisfiability (SAT) and we denote it by QBF. The exponential time hypothesis (ETH) says that we cannot solve 3-SAT in $2^{o(n)}$ time. Similarly, a simplified form of the strong exponential time hypothesis (SETH) says that we cannot solve CNF-SAT in $2^{(1-\varepsilon)\cdot n}$ time. We consider corresponding hypotheses for QBF. In particular, QBF-ETH denotes the hypothesis that we cannot solve QBF in $2^{o(n)}$ time and QBF-SETH denotes the hypothesis that we cannot solve QBF in $2^{(1-\varepsilon)\cdot n}$ time.

Since QBF $\in$ DSPACE$^b(n + O(\log(n)))$, one could apply the approach from the preceding theorem to get the following corollaries.

Corollary 7.19. *If there exists k such that k-IE$_{\mathcal{D}}$ is solvable in $O(n^{k-\varepsilon})$ time, then QBF-SETH is false.*

Corollary 7.20. *If there exist functions f and g such that $g \in o(n)$ and there exists a uniform algorithm for solving (k-IE$_{\mathcal{D}}$) in $f(k)\cdot n^{g(k)}$ time, then QBF-ETH is false.*

7.2.3 SAT-Hardness

Work from [12, 15] showed that more efficient algorithms for $W[1]$-complete problems would refute the exponential time hypothesis and collapse parts of the W-hierarchy. Further, work from [54] showed that slightly more efficient algorithms for $W[2]$-complete problems would refute the strong exponential time hypothesis.

In Chapter 6, we showed that the intersection non-emptiness problem for tree shaped DFA's is $W[1]$-complete. We present results to show similar implications for the existence of more efficient algorithms for this intersection problem.

Theorem 7.21. *If $(2,2)$-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 $(2,2)$-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. □

Theorem 7.22. *If there exists k such that (c,k)-IE$_{\mathcal{TD}}$ is solvable in $O(n^{k-\varepsilon})$ time where $c = k$, then SETH is false.*

Proof. In a similar manner as Theorem 7.21, we reduce CNF-SAT to the intersection problem[1]. 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. □

[1]Alternatively, one could reduce k-dominating set to the intersection problem where $c = k$ and them apply the conditional lower bound from [54]. Further, this alternative approach would show that the intersection problem where $c = k$ is $W[2]$-hard.

Theorem 7.23. *If (c,k)-IE$_{\mathcal{TD}}$ is solvable in $O(n^{\delta})$ time, then c-SAT is solvable in $\mathsf{poly}(n) \cdot 2^{\frac{\delta n}{k}}$ time.*

Sketch of proof. In Theorem 7.22, it was shown how to reduce CNF-SAT to (c,k)-IE$_{\mathcal{TD}}$ where $c = k$. 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. □

By applying the equivalence between the intersection problem and the clique problem from Chapter 6, the next theorem follows from a conditional lower bound for the clique problem from [12].

Theorem 7.24. *If there exist functions f and g such that $g \in o(n)$ and there exists a uniform algorithm for solving $((2,k)$-IE$_{\mathcal{TD}})$ in $f(k) \cdot n^{g(k)}$ time, then ETH is false.*

8

Conclusion

8.1 Summary of Results

By exploiting the dual nature of the intersection non-emptiness problem as a constraint satisfaction and graph reachability problem, we were able to show parameterized equivalences with classical Turing machine acceptance problems. Further, we showed how intersection non-emptiness for different kinds of automata classify different kinds of computational machines.

Following from this classification, we were able to show parameterized completeness results for classes $W[1]$, $W[\text{NL}]$, XL, XNL, XP, and XEXP. In addition, we showed that solving intersection non-emptiness problems more space or time efficiently in some cases is either impossible, would lead to major complexity class separation results, or would refute well known hypotheses about classical complete problems.

In the following table, we list the primary equivalences and completeness results that have been presented in this work.[1]

[1] A cell is denoted with N/A if no well known machine model or parameterized complexity class appropriately represents the intersection non-emptiness problem.

Main Complexity Results for Intersection Non-Emptiness Problems			
Type of Automata	Intersection Problem	Acceptance Problem	Parameterized Complexity
k Tree Shaped DFA's	(k-IE$_{\mathcal{TD}}$)	N/A	W[1]-complete
1 Acyclic DFA and k Tree Shaped DFA's	(k-IE$_{1\mathcal{AC}+\mathcal{TD}}$)	($N^{(S,G)}_{(\log,k\log)}$)	W[NL]-complete
k Acyclic DFA's	(k-IE$_{\mathcal{AC}}$)	($N^{(T,S)}_{(n,k\log)}$)	N/A
k Symmetric Finite Automata	(k-IE$_{\mathcal{S}}$)	($D^{S}_{k\log}$)	XL-complete
k Deterministic Finite Automata	(k-IE$_{\mathcal{D}}$)	($N^{S}_{k\log}$)	XNL-complete
k Acyclic Tree Automata	(k-IE$_{\mathcal{AC}\mathcal{T}}$)	($A^{(T,S)}_{(n,k\log)}$)	N/A
k Tree Automata	(k-IE$_{\mathcal{T}}$)	($A^{S}_{k\log}$)	XP-complete
1 Pushdown Automaton and k DFA's	(k-IE$_{1\mathcal{P}+\mathcal{D}}$)	($Aux^{S}_{k\log}$)	XP-complete
1 Pushdown Tree Automaton and k TA's	(k-IE$_{1\mathcal{PT}+\mathcal{T}}$)	($AltAux^{S}_{k\log}$)	XEXP-complete

8.2 Further Work

This thesis includes the author's primary contributions on intersection non-emptiness problems. Many partial results have been excluded because they are incomplete or of an unrelated form. With plans to continue working in this area we discuss further directions.

Applications to Database Theory: Join problems and intersection non-emptiness problems seem to have a natural correspondence. In particular, a table corresponds with a finite automaton and the join of tables corresponds with a product of automata. Using this correspondence, the non-empty natural join problem for k tables with incomplete information can be related to the intersection non-emptiness for k tree shaped automata. In addition, the non-empty join problem for k boolean n-by-log(n) tables can be related to the intersection non-emptiness problem for k unary finite automata.

Hard Problems in Polynomial Time: There are several popular hard problems in polynomial time. For example, boolean matrix multiplication (BMM) and three sum (3SUM). Some of these problems have natural reductions to intersection non-emptiness. For example, the triangle finding problem for a graph with n vertices and m edges can be reduced to intersection non-emptiness for two DFA's where one DFA has n states and the other has m states.

Exact Time Complexity: It's not known if the intersection non-emptiness problem for k DFA's can be solved in less than n^k time. However, the intersection non-emptiness problem for k tree shaped DFA's can be solved in $n^{0.792k}$ time by a reduction to the k-clique problem. Also, although not presented in this work, the intersection non-emptiness problem for k unary DFA's can be solved in $n^{1.4k}$ time by a reduction to triangle finding in a sparse graph. Do there exist any more efficient algorithms?

Parameterized Complexity Lower Bounds: In [36], the authors consider an alternative parameterization of the intersection non-emptiness problem where m denotes the size of the largest DFA, n denotes the size of the second largest DFA, and k denotes the number of DFA's. They showed how the existence of

a uniform algorithm for solving (k-IE$_{\mathcal{D}}$) that runs in $m \cdot n^{o(k)}$ time implies that NL $\subseteq$ DTIME($n^{1+\varepsilon}$) for all $\varepsilon > 0$. In the same spirit as parameterized complexity conditional lower bounds from [13, 12, 14, 15], one can show that if there exist functions f and g such that $g \in o(n)$ and there exists a uniform algorithm for solving (k-IE$_{\mathcal{D}}$) in $f(k) \cdot m^{O(1)} \cdot n^{g(k)}$ time, then FPT = XNL.

Self-Reducibility and Kernelization: The intersection non-emptiness problem has nice self-reducibility properties. In particular, it satisfies the OR-compositional property from [7, 51]. As a result, if intersection non-emptiness has a polynomial kernelization, then work from [23] (see also [19]) suggests that we would have PSPACE $\subseteq$ *co*-NP/*poly*. Moreover, we would get that the polynomial hierarchy collapses to the third level [73]. It further seems that if the kernelization is parameter preserving, then we would get P = PSPACE.

Bibliography

[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] Rajeev Alur and Swarat Chaudhuri. Branching pushdown tree automata. In *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'06, pages 393–404, Berlin, Heidelberg, 2006. Springer-Verlag.

[3] M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-Complete. *Developments in Language Theory*, 2008.

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

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

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

[7] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, December 2009.

[8] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[9] Marco Cesati. The turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67(4):654 – 685, 2003.

[10] Jacques Chabin and Pierre Réty. Visibly pushdown languages and term rewriting. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems: 6th International Symposium, FroCoS 2007 Liverpool, UK, September 10-12, 2007 Proceedings*, pages 252–266, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[11] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

[12] 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.

[13] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Linear fpt reductions and computational lower bounds. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 212–221, New York, NY, USA, 2004. ACM.

[14] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. W-hardness under linear fpt-reductions: Structural properties and further applications. In Lusheng Wang, editor, *Computing and Combinatorics: 11th Annual International Conference, COCOON 2005 Kunming, China, August 16–19, 2005 Proceedings*, pages 975–984, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[15] 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.

[16] D. Chistikov, W. Czerwinski, P. Hofman, M. Pilipczuk, and M. Wehar. Shortest paths in one-counter systems. In *FoSSaCS 2016 (to appear)*, 2016.

[17] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, October 2007.

[18] Stephen A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1):4–18, January 1971.

[19] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[20] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[21] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995.

[22] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[23] Andrew Drucker. New limits to classical and quantum instance compression. In *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, FOCS '12, pages 609–618, Washington, DC, USA, 2012. IEEE Computer Society.

[24] Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space complexity of parameterized problems. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized and Exact Computation: 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings*, pages 206–217, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[25] Henning Fernau, Pinar Heggernes, and Yngve Villanger. A multi-parameter analysis of hard problems on deterministic finite automata. *Journal of Computer and System Sciences*, 81(4):747 – 765, 2015.

[26] Henning Fernau and Andreas Krebs. Problems on finite automata and the exponential time hypothesis. In Yo-Sub Han and Kai Salomaa, editors, *Implementation and Application of Automata: 21st International Conference, CIAA 2016, Seoul, South Korea, July 19-22, 2016, Proceedings*, pages 89–100, Cham, 2016. Springer International Publishing.

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

[28] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for np. *J. Comput. Syst. Sci.*, 77(1):91–106, January 2011.

[29] Zvi Galil. Hierarchies of complete problems. *Acta Informatica*, 6(1):77–88, 1976.

[30] Inène Guessarian. Pushdown tree automata. *Mathematical systems theory*, 16(1):237–263, 1983.

[31] Eitan M. Gurari and Oscar H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22(2):220 – 229, 1981.

[32] Markus Holzer and Pierre McKenzie. Alternating and empty alternating auxiliary stack automata. *Theoretical Computer Science*, 299(1):307 – 326, 2003.

[33] H. B. Hunt, III. On the time and tape complexity of languages I. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 10–19, New York, NY, USA, 1973. ACM.

[34] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, January 1978.

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

[36] 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.

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

[38] Ron Kohavi. Bottom-up induction of oblivious read-once decision graphs: Strengths and limitations. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, pages 613–618, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

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

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

[41] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Pushdown specifications. In Matthias Baaz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning: 9th International Conference, LPAR 2002 Tbilisi, Georgia, October 14–18, 2002 Proceedings*, pages 262–277, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[42] S. L. Torre, P. Madhusudan, and G. Parlato. An infinite automaton characterization of double exponential time. *CSL 2008*, pages 33–48, 2008.

[43] Richard E. Ladner, Richard J. Lipton, and Larry J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13(1):135–155, 1984.

[44] Richard E. Ladner, Larry J. Stockmeyer, and Richard J. Lipton. Alternation bounded auxiliary pushdown automata. *Information and Control*, 62(2):93 – 108, 1984.

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

[46] Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161 – 187, 1982.

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

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

[49] P. Madhusudan and Gennaro Parlato. The tree width of automata with auxiliary storage. *POPL 2011*, 2011.

[50] W. Martens and S. Vansummeren. Automata and logic on trees: Algorithms. ESSLLI 2007, 2007.

[51] Neeldhara Misra, Venkatesh Raman, and Saket Saurabh. Lower bounds on kernelization. *Discrete Optimization*, 8(1):110 – 128, 2011. Parameterized Complexity of Discrete Optimization.

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

[53] Johannes Osterholzer. Complexity of uniform membership of context-free tree grammars. In Andreas Maletti, editor, *Algebraic Informatics: 6th International Conference, CAI 2015, Stuttgart, Germany, September 1-4, 2015. Proceedings*, pages 176–188, Cham, 2015. Springer International Publishing.

[54] 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.

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

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

[57] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, September 2008.

[58] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM.

[59] 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.

[60] Shinichi Tanaka and Takumi Kasai. The emptiness problem for indexed language is exponential-time complete. *Systems and Computers in Japan*, 17(9):29–37, 1986.

[61] H. Todd Wareham. The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Shen Yu and Andrei Păun, editors, *Implementation and Application of Automata: 5th International Conference, CIAA 2000 London, Ontario, Canada, July 24–25, 2000 Revised Papers*, pages 302–310, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[62] S. L. Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. *LICS 2007*, pages 161–170, 2007.

[63] L. G. Valiant. Decision procedures for families of deterministic pushdown automata. Technical report, University of Warwick, Coventry, UK, 1973.

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

[65] Margus Veanes. On computational complexity of basic decision problems of finite tree automata. *UPMAIL Technical Report 133*, 1997.

[66] Igor Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234 – 263, 2001.

[67] Michael Wehar. Intersection emptiness for finite automata. Honors thesis, Carnegie Mellon University, 2012.

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

[69] Michael Wehar. Intersection non-emptiness for tree shaped finite automata. *Unpublished*, 2016.

[70] Michael Wehar. Solving the inversion problem. Private communication, September 2016.

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

[72] 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.

[73] Chee K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26(3):287 – 300, 1983.