# On The Fine Grained Complexity of Multi-Stack Reachability

**Michael Wehar**
Bryn Mawr College

Infinity Workshop 2026

# Outline

# Complexity Classes

**The following complexity classes are known to be equivalent:**

- Polynomial Time : $\texttt{PTIME} = \bigcup_k \texttt{DTIME}(n^k)$
- Alternationg Log Space : $\texttt{AL} = \texttt{ASPACE}(\log(n))$
- Auxiliary Log Space : $\texttt{AuxL} = \texttt{AuxSPACE}(\log(n))$

**These equivalences were shown in:**

- [Cook 1971] $\texttt{PTIME} = \texttt{AuxL}$
- [Chandra, Kozen, Stockmeyer 1981] $\texttt{PTIME} = \texttt{AL}$

# Alternating Machines

**Recall:**

- Alternating TM's (ATM's) have existential and universal states.
- A configuration consists of a state, tape head positions, and tape contents represented as strings.
- A computation is a tree of configurations where the root is an initial configuration, the leaves are halting configurations, and every parent transitions to every one of its children.
- Existential and universal configurations are those corresponding with existential and universal states, respectively.
- A computation is valid if all existential configurations have a single child and all universal configurations have all configurations it could transition to as children.

# Auxiliary Pushdown Automata

**Recall:**

- Auxiliary Pushdown Automata (AuxPDA's) are Turing machines that are given a stack in addition to a read / write work tape.
- We will focus on deterministic AuxPDA's as there is an equivalence with non-deterministic variants.
- Space bounds apply to the work tape while the stack is unrestricted.
- For an AuxPDA with $m$ states, an input string of length $n$, and at most $k \cdot \log(n)$ binary work tape cells, we have essentially $\gamma(m, n) = m \cdot n^{k+1}$ possible configurations ignoring the stack.
- One could bound the stack height and computation length in terms of $\gamma(m, n)$ for halting AuxPDA's.

# Tapes and Alphabets: Part 1

- When defining complexity classes, we often allow for a variable number of tapes and variable size tape alphabets.
- This is problematic for space complexity classes because there is a clear difference between $c_1 \cdot s(n)$ bits of memory and $c_2 \cdot s(n)$ bits of memory when $c_1 < c_2$.
- If we count the number of possible machine configurations, the coefficients $c_1$ and $c_2$ move to the exponent resulting in a substantial difference in the number of configurations.
- Therefore, we need to fix the number of tapes and tape alphabets.

# Tapes and Alphabets: Part 2

- We consider Turing machines that have a two-way read-only input tape and a two-way read / write tape.
- The tape alphabets are binary aside from a constant number of occurrences of a delimiter symbol.
- We write $\texttt{ASPACE}_b$ and $\texttt{AuxSPACE}_b$ to denote complexity measured relative to these machines.
- As a result, we have $\texttt{ASPACE}(\log(n)) = \texttt{ASPACE}_b(O(\log(n)))$ and $\texttt{AuxSPACE}(\log(n)) = \texttt{AuxSPACE}_b(O(\log(n)))$.

# Parameterized Complexity

- Looking at the classic arguments we observe that the arguments preserve the parameter $k$. [ICALP 2015]
- That is, there exist constants $c_1$ and $c_2$ such that for every $k$:
  - $\texttt{DTIME}(n^k) \subseteq \texttt{ASPACE}_b(c_1 \cdot k \cdot \log(n))$
  - $\texttt{DTIME}(n^k) \subseteq \texttt{AuxSPACE}_b(c_2 \cdot k \cdot \log(n))$
- Similarly, there exist constants $d_1$ and $d_2$ such that for every $k$:
  - $\texttt{ASPACE}_b(k \cdot \log(n)) \subseteq \texttt{DTIME}(n^{d_1 \cdot k})$
  - $\texttt{AuxSPACE}_b(k \cdot \log(n)) \subseteq \texttt{DTIME}(n^{d_2 \cdot k})$

# Unconditional Time Lower Bounds

- By combining these result together, we have that for every $k$:
  - $\texttt{DTIME}(n^k) \subseteq \texttt{ASPACE}_b(c_1 \cdot k \cdot \log(n)) \subseteq \texttt{DTIME}(n^{c_1 \cdot d_1 \cdot k})$
  - $\texttt{DTIME}(n^k) \subseteq \texttt{AuxSPACE}_b(c_2 \cdot k \cdot \log(n)) \subseteq \texttt{DTIME}(n^{c_2 \cdot d_2 \cdot k})$
- Furthermore, for every $k$, there exist languages in $\texttt{ASPACE}_b(k \cdot \log(n))$ and $\texttt{AuxSPACE}_b(k \cdot \log(n))$ that are not in $\texttt{DTIME}(n^{\frac{k}{c_1} - \varepsilon})$ and $\texttt{DTIME}(n^{\frac{k}{c_2} - \varepsilon})$, respectfully, for all $\varepsilon > 0$.
- Otherwise, if say $\texttt{ASPACE}_b(k \cdot \log(n)) \subseteq \texttt{DTIME}(n^{\frac{k}{c_1} - \varepsilon})$, then by padding $\texttt{DTIME}(n^k) \subseteq \texttt{ASPACE}_b(c_1 \cdot k \cdot \log(n)) \subseteq \texttt{DTIME}(n^{k - c_1 \cdot \varepsilon})$ which would violate the deterministic time hierarchy theorem.

# Outline

# Three Kinds of Automata

**We will discuss three kinds of automata:**

- Tree Automata
- Pushdown Automata
- Multi-Stack Pushdown Automata

# Top-Down Tree Automata

- Tree automata are a generalization of finite automata that read in labeled trees instead of strings.
- Whenever a node in the tree has $c$ children, the automaton will transition from one state to $c$ states, one for each child.
- The automaton will independently carry on the evaluation down each branch of the tree with the respective state.
- The non-emptiness problem asks whether there exists a tree that is accepted by the tree automaton. It is solvable efficiently in terms of the size of the tree automaton.
- It's like BFS, but with marking the children of directed hyperedges.

# Pushdown Automata

- A pushdown automaton (PDA) augments a finite automaton by adding a stack as memory. For a PDA $P$, we let $\mathcal{L}(P)$ denote the language that $P$ recognizes. PDA's recognize context-free languages.
- Given a pushdown automaton $P$ and a DFA $D$ we can use the Cartesian product construction to build a PDA $P'$ such that $\mathcal{L}(P') = \mathcal{L}(P) \cap \mathcal{L}(D)$.
- The non-emptiness problem for PDA's is deterministically solvable in $O(n^3)$ time where $n$ is the number of states.
- [Chistikov, Majumdar, Schepper 2022] showed that we can solve non-emptiness more efficiently with non-determinism.

# Multi-Stack Pushdown Automata: Part 1

- A two-stack automaton can simulate a Turing machine because two stacks can be used to simulate a tape.
- However, if we add constraints on when the stacks can be pushed to and popped from, non-emptiness becomes decidable.
- Consider a (one-way) non-deterministic automaton with $k$ stacks where all stacks can push, but only one designated stack can pop.
- We can change which stack is designated for popping only a constant number of times. Each change is called a phase switch.
- Such machines are called Multi-Stack Pushdown Automata with bounded phase switching.

# Multi-Stack Pushdown Automata: Part 2

- The non-emptiness problem for Multi-Stack Pushdown Automata with $k$-phase switches is in $\texttt{DTIME}(n^{c \cdot 2^k})$ for some constant $c$ independent of $k$. [Madhusudan, Parlato 2011]
- The algorithm involves using monadic second-order logic definable graphs of bounded tree width to represent accepting computations of the mulit-stack pushdown automaton.
- They provide an algorithm for checking if well-behaved graphs of tree width at most $t$ exist in $O(n^{O(t)})$ time where $n$ is the number of states from the multi-stack pushdown automaton.
- It is shown that an accepting computation exists if and only if there exists such a graph with tree width $O(2^k)$.

# Outline

# Lower Bounds for Three Problems

**We will discuss lower bounds for three problems:**

- Intersection Non-Emptiness for Tree Automata
  - By verifying computations for space bounded ATM's.
- Intersection Non-Emptiness for One PDA and many DFA's
  - By verifying computations for space bounded AuxPDA's.
- Non-Emptiness for Multi-Stack Pushdown Automata
  - By simulating the evaluation of one PDA and exponentially many DFA's on the same input string.

# Intersection Non-Emptiness for Tree Automata

- Given a finite list of Tree Automata, determining whether there exists a tree that simultaneously is accepted by all of the tree automata is EXPTIME-complete. [Veanes 1997]
- Essentially, we can use Tree Automata to verify computations for alternating Turing machines. Each tree automaton is responsible for keeping track of $\log(n)$ bits of the work tape.
- In particular, there exists a constant $t$ such that for all $k$, $\mathtt{ASPACE}_b(k \cdot \log(n))$ computations can be verified by families of $k$ Tree automata each with at most $n^t$ states.
- Since $\mathtt{DTIME}(n^k) \subseteq \mathtt{ASPACE}_b(c_1 \cdot k \cdot \log(n))$, we have that we cannot solve this problem for $k$ Tree automata in $\mathtt{DTIME}(n^{\frac{k}{t \cdot c_1} - \varepsilon})$.

# Intersection Non-Emptiness for one PDA and DFA's

- Given one Pushdown Automaton and a finite list DFA's, determining whether there exists a string that simultaneously is accepted by all of the automata is EXPTIME-complete.
- Essentially, we can use the PDA and the DFA's together to verify computations for AuxPDA's where the one PDA checks the stack usage and the DFA's check the tape usage. Each DFA keeps track of $\log(n)$ bits of the work tape.
- In particular, there exists a constant $p$ such that for all $k$, $\mathtt{AuxSPACE}_b(k \cdot \log(n))$ computations can be verified by families of one PDA and $k$ DFA's each with at most $n^p$ states.

# One PDA and DFA's: Part 2

- Since $\texttt{DTIME}(n^k) \subseteq \texttt{AuxSPACE}_b(c_2 \cdot k \cdot \log(n))$, we have that we cannot solve this problem for one PDA and $k$ DFA's in $\texttt{DTIME}(n^{\frac{k}{p \cdot c_2} - \varepsilon})$.
- In fact, the one PDA can actually be made fixed. We can simply vary the DFA's without the need to change the stack verification process.

# Non-Emptiness for Multi-Stack PDA's: Part 1

- With $k$ phase switches, we can use the stacks to guess a string $w$ and make $O(2^k)$ copies of $w$.
- Each time we read a copy of $w$ we simulate a DFA on $w$. Each time we read $w^R$ we simulate the NFA obtained by reversing the DFA.
- As a result, we are able to simulate $2^k - 1 = 1 + 2 + 2^2 + ... + 2^{k-1}$ DFA's on input $w$.
- We simulate the PDA on input $w$, but we can only do this once.
- When we read the input $w$ for the first time it is from the input tape. All future times we do so by popping $w$ (or $w^R$) from a stack.

# Non-Emptiness for Multi-Stack PDA's: Part 2

- As a result, we have reduced intersecting one PDA and $O(2^k)$ DFA's to solving non-emptiness for multi-stack pushdown automata with $k$-phase switches.
- The prior lower bound translates to a lower bound for non-emptiness for multi-stack pushdown automata with $k$-phase switches.
- Therefore, there exists a constant $s$ such that this problem is not solvable in $\texttt{DTIME}(n^{s \cdot 2^k - \varepsilon})$ for all $k$ and all $\varepsilon > 0$.
- Similarly, we can prove a lower bound for non-emptiness for a dual class of multi-stack PDA's where we can pop from any stack, but only push to a designated stack.

# Outline

# Optimizing Constants: $c_1$, $c_2$

- Recall that:
  - $\texttt{DTIME}(n^k) \subseteq \texttt{ASPACE}_b(c_1 \cdot k \cdot \log(n))$
  - $\texttt{DTIME}(n^k) \subseteq \texttt{AuxSPACE}_b(c_2 \cdot k \cdot \log(n))$
- But, how small can we make $c_1$ and $c_2$? Can we optimize these values to compute precise time complexity lower bounds?
- **Answer:** Yes, I think we can! I will attempt to do so in the following and make some proposed claims that **require** further verification.

# Revisiting PTIME $\subseteq$ AL: Part 1

- This argument works as follows. Given a Turing machine $M$ that runs for at most $n^k$ time on inputs of length $n$.
- We construct an alternating Turing machine that will essentially verify that specific bits on the work tape are correct at specific times.
- We have a few predicates:
  - $W(t, i, b)$ : at time step $t$, the work tape has bit $b$ at position $i$
  - $Pr(t, i)$ : at time step $t$, the input tape head is at position $i$
  - $Pw(t, i)$ : at time step $t$, the work tape head is at position $i$
  - $S(t, s)$ : at time step $t$, the state is $s$

# Revisiting PTIME $\subseteq$ AL: Part 2

- We describe how to verify correctness of a predicate on input values:
  - **W(t + 1, i, b)** : [ $Pw(t, i')$ where $i' \neq i$ and $W(t, i, b)$ ] or [ $\exists\ i_r,\ s,\ b'$ s.t. $Pr(t, i_r) \wedge Pw(t, i) \wedge S(t, s) \wedge W(t, i, b')$ and there is a transition corresponding with these values in $M$ ].
  - **Pr(t + 1, i)** : Similar to "right" part of previous predicate.
  - **Pw(t + 1, i)** : Similar to previous predicate.
  - **S(t, s)** : Similar to previous predicate.
- Notice that we essentially just need to store three values that are $k \log(n)$ bits and a constant number of values that are $\log(n)$ bits.
- Therefore, the proposed space complexity is $(3k + O(1)) \cdot \log(n)$.

# Optimizing Constants: $t$, $p$

- Recall that:
  - The $k$ tree automata each have size $n^t$ for verifying $\texttt{ASPACE}_b(k \cdot \log(n))$ computations
  - The $k$ DFA's each have size $n^p$ for verifying $\texttt{ASPACE}_b(k \cdot \log(n))$ computations
- But, how small can we make $t$ and $p$? Can we optimize these values to compute precise time complexity lower bounds?
- **Answer:** Yes, $t, p = 1 + o(1)$! We have an arXiv manuscript showing how to perform the verification with automata each of size $n^{1+o(1)}$ with one extra automaton. We then apply a lower bound improvement lemma to obtain the desired result for one less automaton.

# Outline

# Conclusion

- Non-Emptiness Problems and Intersection Non-Emptiness problems have unconditional time complexity lower bounds.
- These lower bounds are a result of simulating Turing machine models.
- Ultimately, the lower bounds are a result of the deterministic time hierarchy theorem.
- By applying a fine-grained perspective, we can compute more precise lower bounds for all $k$ and $\varepsilon > 0$. We propose the following lower bounds that I am working to further verify:
  - Intersection non-emptiness for $k$ tree automata $\notin \texttt{DTIME}(n^{\frac{k}{3}-\varepsilon})$
  - Intersection non-emptiness for One PDA and $k$ DFA's $\notin \texttt{DTIME}(n^{\frac{k}{3}-\varepsilon})$

# Thank You!

Questions or Comments?