Limited Nondeterminism of Input-Driven Pushdown Automata: Decidability and Complexity

Yo-Sub Han, Sang-Ki Ko, Kai Salomaa

Limited Nondeterminism of Input-Driven Pushdown Automata: Decidability and Complexity

Yo-Sub Han[1], Sang-Ki Ko[2], and Kai Salomaa[3]

[1] Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
emmous@yonsei.ac.kr
[2] AI Research Center, Korea Electronics Technology Institute
Gyeonggi-do, Republic of Korea
sangkiko@keti.re.kr
[3] School of Computing, Queen's University, Kingston
Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. We study the decidability and computational complexity for several decision problems related to limited nondeterminism of finite-state automata equipped with a pushdown stack. Ambiguity and tree width are two measures of nondeterminism considered in the literature. As a main result, we prove that the problem of deciding whether or not the tree width of a nondeterministic pushdown automaton is finite is decidable in polynomial time. We also prove that the k-tree width problem for nondeterministic input-driven pushdown automata (respectively, nondeterministic finite automata) is complete for exponential time (respectively, for polynomial space).

Keywords: nondeterminism, tree width, ambiguity, input-driven pushdown automata

1 Introduction

Nondeterminism plays a fundamental role in automata and formal language theory. There have been various ways to quantify the nondeterminism of automaton models [7, 8, 11, 16, 17]. For instance, the expressive power of finite automata is the same even if we allow the use of nondeterminism since both deterministic (DFA) and and nondeterministic finite automata (NFA) accept the class of regular languages. However, the descriptional complexity of minimal DFAs and NFAs accepting the same regular language differs significantly as it is well-known that the determinization of NFAs causes an exponential blow-up in size of the resulting DFAs in the worst-case.

On the other hand, for some models such as pushdown automata, the expressive power of the automata is strictly increased if we allow the use of nondeterminism. In other words, there exist (context-free) languages that can be

described by a nondeterministic pushdown automaton (NPDA) and not by a determinististic PDA (DPDA).

The *ambiguity* of a nondeterministic machine means the number of accepting computations on an input string. An NFA is said to be an *unambiguous finite-state automaton (UFA)* if it has a unique accepting computation on every accepted string. Obviously, every DFA is an UFA. Similarly, there are *finitely ambiguous automata* where the number of accepting computations is bounded by a constant. For the remaining automata that have an unbounded number of accepting computations, we can further distinguish between *polynomially ambiguous* and *exponentially ambiguous* depending on the relationship between the length of the string and the number of accepting computations on the string [13]. Chan and Ibarra [5] proved that the problem of deciding whether or not the ambiguity of a given NFA is bounded by a given integer k is PSPACE-complete. They also showed that the problem of deciding whether or not the ambiguity of an NFA is bounded by a constant can be solved in polynomial space by applying a matrix algorithm. Later, Weber and Seidl [20] presented a polynomial time algorithm for the problem by observing a simple criterion which characterizes the infinite degree of ambiguity of an NFA.

There is another approach for quantifying the nondeterminism of automata models called *leaf size* [11, 3], *path size* [16, 15], or *tree width* [17]. The tree width means the total number of (complete and incomplete) computations on an input string regardless of the acceptance of the string. Palioudakis et al. [17] considered NFAs having finite tree width where the computation on any input string has a constant number of branches. They gave effective characterizations of such NFAs and a tight bound for the determinization as a function of the tree width and the size of an NFA. They also revealed that the problem of deciding whether or not a given NFA has finite tree width is decidable in polynomial time.

Okhotin and Salomaa [16] studied the descriptional complexity of determinizing an NIDPDA that has limited nondeterminism. They called an NIDPDA with at most k computations on any input a *k-path NIDPDA*. In other words, the tree width of the NIDPDA is bounded by k. It is known that the size of the smallest DIDPDA obtained from an NIDPDA of size n is $2^{\Theta(n^2)}$ [2] but in the case of a k-path NIDPDA, the determinization yields a DIDPDA of size at most $\Theta(n^k)$. They also provided an algorithm for deciding whether or not a given NIDPDA has the k-path property and showed that the problem is P-complete for a fixed k. Finally, Caralp et al. [4] studied the decision problems regarding the ambiguity of NIDPDAs by investigating an interesting extension of IDPDAs. They proposed an extension of IDPDAs by means of positive integer weights associated with transitions. Then, they define the *multiplicity* of an NIDPDA as the supremum of multiplicities over all possible input strings, which is defined as the degree of ambiguity here. They proved that the *k-boundedness problem* (the problem of deciding whether or not the ambiguity is bounded by k) is EXPTIME-complete (complete for exponential time) and the *finiteness problem* (the problem of deciding whether or not the ambiguity is bounded by a constant) is solvable in polynomial time.

We study several unsolved cases of the complexity and decidability questions related to limited nondeterminism for computational models such as NFAs, NIDPDAs, and NPDAs. For instance, the decidability of the problem of deciding whether or not a given NIDPDA has finite tree width (i.e., finite path size) has been left open in [16]. We complete the complexity and decidability landscape presented in Table 1 regarding the decision problems about the ambiguity and the tree width of NFAs, NIDPDAs, and NPDAs. In Section 2, we give basic definitions and notations that are used throughout the paper. Section 3 provides the results on the decision problems for the tree width and ambiguity of NPDAs. In Section 4, we present the new results on the decision problems for the tree width of NIDPDAs and NFAs. Lastly, Section 5 concludes the paper.

Problem	For NPDAs		For NIDPDAs	For NFAs
k-ambiguity	**Undecidable (Prop. 4)**		EXPTIME-c [4]	PSPACE-c [5]
k-tree width	$k \geq 3$	Undecidable [16]	EXPTIME-**c**	PSPACE-**c**
	$k < 3$	P **(Prop. 3)**	**(Theorem 5)**	**(Theorem 6)**
Finite tree width	P **(Theorem 2)**		P **(Theorem 2)**	P [17]
Finite ambiguity	Undecidable [9, 21]		P [4]	P [20]

Table 1. The complexity landscape of decision problems regarding the nondeterminism in NFAs, NIDPDAs, and NPDAs. Our results are written in bold. We assume that the integer k is given as part of input and represented in unary notation.

2 Preliminaries

We briefly present definitions and notations used throughout the paper. The reader may refer to the textbooks [10, 19, 21] for complete knowledge of automata and formal language theory.

Let Σ be a finite alphabet and Σ^* be a set of all strings over Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language, the symbol λ denotes the null string and Σ^+ denotes $\Sigma^* \setminus \{\lambda\}$.

A *nondeterministic finite automaton* (NFA) is specified by a quintuple $A = (\Sigma, Q, q_0, F, \delta)$ where Σ is a finite alphabet, Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ is the set of final states and δ is a multi-valued transition function from $Q \times \Sigma$ into 2^Q. The automaton A is *deterministic* (a DFA) if δ is a (single-valued) function $Q \times \Sigma \to Q$. It is well known that the NFAs and DFAs all recognize the regular languages [18, 19, 21].

A *nondeterministic pushdown automaton* (NPDA) is specified by a tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite stack alphabet, $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{Q \times \Gamma^{\leq 2}}$ is a transition function, $q_0 \in Q$ is the start state, Z_0 is the initial stack symbol and $F \subseteq Q$ is the set of final states. Our definition restricts that each transition of P has at most two stack symbols, that is, each transition can push or pop at most one symbol. We use $|\delta|$ to denote the number of transitions in δ.

A *configuration* of an NPDA P is a triple (q, w, v), where $q \in Q$ is the current state, $w \in \Sigma^*$ is the remaining input, and $v \in \Gamma^*$ is the contents on the stack. Denote the set of configurations of P by $C(P)$ and we define the single step computation relation as $\vdash_P \subseteq C(P) \times C(P)$. The language $L(P)$ is the set of strings accepted by P.

Consider $q, p \in Q$, $b \in (\Sigma \cup \lambda)$, $X \in \Gamma$ and $u \in \Gamma^*$. If $(p, u) \in \delta(q, b, X)$, we say that $(q, b, X) \to (p, u)$ is a transition of P, and it is called a λ-transition if $b = \lambda$ (i.e., a λ-transition does not consume an input symbol). For $q \in Q$, $b \in (\Sigma \cup \lambda)$ and $X \in \Gamma$, a transition $(q, b, X) \to (p, u)$ is *nondeterministic* if

1. for some $(p', u') \neq (p, u)$, $(p', u') \in \delta(q, b, X)$, or,
2. b is an element of Σ and $\delta(q, \lambda, X) \neq \emptyset$, or,
3. $b = \lambda$ and there exists $c \in \Sigma$ such that $\delta(q, c, X) \neq \emptyset$.

A transition $(q, b, X) \to (p, u)$ is nondeterministic if, roughly speaking, the tuple (q, b, X) allows the NPDA to make also a different transition.

The question whether a λ-transition involves a nondeterministic step may depend on what is the next input symbol and we need to be a little more precise in the definition. A λ-transition $(q, \lambda, X) \to (p, u)$ is said to be *inherently nondeterministic* if there exist $(p', u') \neq (p, u)$ such that $(p', u') \in \delta(q, \lambda, X)$ (i.e., the case 1. holds). A λ-transition $(q, \lambda, X) \to (p, u)$ which is not inherently nondeterministic, is said to be *Z-nondeterministic*, $\emptyset \neq Z \subseteq \Sigma$, if the set Z consists of all alphabet symbols $c \in \Sigma$ such that $\delta(q, c, X) \neq \emptyset$. Note that applying an inherently nondeterministic λ-transition always involves a nondeterministic step. Applying a Z-nondeterministic ($Z \subseteq \Sigma$) λ-transition involves a nondeterministic step only if the next input symbol belongs to Z.

A *nondeterministic input-driven pushdown automaton* (NIDPDA) [1, 14, 16] is a restricted version of an NPDA, where the input alphabet Σ consists of three disjoint classes, Σ_c, Σ_r, and Σ_l. Namely, $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$. The class where the input symbol belongs to determines the type of stack operation: The automaton always pushes a symbol onto the stack when it reads a *call symbol* in Σ_c. If the input symbol is a *return symbol* in Σ_r, the automaton pops a symbol from the stack. Finally, the automaton neither uses the stack nor even examines the content of the stack for the *local symbols* in Σ_l. Formally, the input alphabet is defined as $\widetilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_l)$, where three components are finite disjoint sets.

An NIDPDA is formally defined by a tuple $A = (\widetilde{\Sigma}, \Gamma, Q, q_0, F, \delta_c, \delta_r, \delta_l)$, where $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$ is the input alphabet, Γ is the finite set of stack symbols, Q is the finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final states, $\delta_c : Q \times \Sigma_c \to 2^{Q \times \Gamma}$ is the transition function for the push operations, $\delta_r : Q \times (\Gamma \cup \{\bot\}) \times \Sigma_r \to 2^Q$ is the transition function for the pop operations, and $\delta_l : Q \times \Sigma_l \to 2^Q$ is the local transition function. We use $\bot \notin \Gamma$ to denote the top of an empty stack. The single step transition relation $\vdash_A$ of an NIDPDA A is described as follows:

- **Push operation:** $(q, aw, v) \vdash_A (q', w, \gamma v)$ for all $a \in \Sigma_c, (q', \gamma) \in \delta_c(q, a), \gamma \in \Gamma, w \in \Sigma^*$ and $v \in \Gamma^*$.

- **Pop operation:** $(q, aw, \gamma v) \vdash_A (q', w, v)$ for all $a \in \Sigma_r, q' \in \delta_r(q, \gamma, a), \gamma \in \Gamma, w \in \Sigma^*$ and $v \in \Gamma^*$; furthermore, $(q, aw, \lambda) \vdash_A (q', w, \lambda)$, for all $a \in \Sigma_r, q' \in \delta_r(q, \bot, a)$ and $w \in \Sigma^*$.
- **Local operation:** $(q, aw, v) \vdash_A (q', w, v)$, for all $a \in \Sigma_l, q' \in \delta_l(q, a), w \in \Sigma^*$ and $v \in \Gamma^*$.

An *initial configuration* of an NIDPDA $A = (\widetilde{\Sigma}, \Gamma, Q, s, F, \delta_c, \delta_r, \delta_l)$ is (s, w, λ), where s is the start state and w is an input word. an NIDPDA accepts a word if A arrives at a final state by reading the word from the initial configuration. Formally, we write the language recognized by A as

$$L(A) = \{w \in \Sigma^* \mid (s, w, \lambda) \vdash_A^* (q, \lambda, v) \text{ for some } q \in F, v \in \Gamma^*\}.$$

We call the languages recognized by NIDPDAs the *input-driven pushdown languages*. The class of input-driven pushdown languages is a strict subclass of deterministic context-free languages and a strict superclass of regular languages. While many closure properties such as complement and intersection do not hold for context-free languages, input-driven pushdown languages are closed under most operations including other basic operations such as concatenation, union, and Kleene-star.

Below we use NPDA to define various nondeterminism measures since the same definition holds for NIDPDA and NFA as they are special cases of NPDA.

Computation trees and tree width [17] (a.k.a. path size [15]). Let us consider an NPDA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ and be $C = (q, w, y)$, where $q \in Q$ is a state, $w \in \Sigma^*$ is the remaining input, and $y \in \Gamma^*$ is the current stack contents, be a configuration of A. Then, the initial configuration of the NPDA A on a string w is $C_0 = (q_0, w, Z_0)$. If a configuration C in one computation step of A yields configurations $C_1, \ldots, C_m, m \geq 0$, we denote it by $C \vdash_A (C_1, \ldots, C_m)$. For example, the successor configurations of a configuration $C = (q, bu, Xs)$ are obtained by applying m different transitions applicable to triple (q, b, X) and also by applying λ-transitions applicable to (q, λ, X). Now the *computation tree* of A on input w is defined as follows:

1. the root node is labelled by the initial configuration C_0;
2. if a node v is labelled by a configuration C and $C \vdash_A (C_1, \ldots, C_m), m \geq 0$, then the node v has m children that are labelled, respectively, by $C_1, \ldots, C_m$ in the computation tree;
3. if a node v is labelled by configuration C and no computation step is defined in C, then the node v is a leaf node.

Intuitively, the tree width measure counts the number of partial computations on a given input and for NFAs it is normally defined in terms of the number of leaves of the computation tree. However, since an NPDA A may have infinite cycles caused by λ-transitions we define the *tree width of* A on an input w, $\mathrm{tw}_A(w)$, as the maximal number of pairwise independent nodes in the computation tree of A on w. (For automaton models that do not allow infinite cycles,

such as NFAs or NIDPDAs, this value coincides with the number of leaves of the computation tree.)

The (maximum) tree width of A on strings of length m is defined as $\mathrm{tw}_A(m) = \max\{\mathrm{tw}_A(w) \mid w \in \Sigma^m\}$. The tree width of A is finite if the values $\mathrm{tw}_A(m)$, where $m \in \mathbb{N}$, are bounded and in that case we denote $\mathrm{tw}_A^{\sup} = \sup_{m\in\mathbb{N}} \mathrm{tw}_A(m)$.

Note that we can define the tree width of any other nondeterministic devices exactly in the same way. (For devices without a pushdown stack, we just omit the third component.) We say that an NPDA A has tree width k if the tree width of A on any string is at most k and there exists a string w where the tree width of A on w is k, that is, if $\mathrm{tw}_A^{\sup} = k$.

Ambiguity [5, 20]. An automaton (which can be either an NFA, an NPDA, or an NIDPDA) A is *unambiguous* if any string has at most one accepting computation. The degree of ambiguity of A on a string w is the number of accepting computations of A on w and denoted by $\mathrm{da}_A(w)$. More formally, $\mathrm{da}_A(w)$ is the number of leaves of the computation tree of A on w that are labeled by accepting configurations.

The *degree of ambiguity* of A on strings of length m is defined as $\mathrm{da}_A(m) = \max\{\mathrm{da}_A(w) \mid w \in \Sigma^m\}$. The degree of ambiguity of A is finite if the values $\mathrm{da}_A(m)$, where $m \in \mathbb{N}$, are bounded as follows: $\mathrm{da}_A^{\sup} = \sup_{m\in\mathbb{N}} \mathrm{da}_A(m)$.

The automaton A is *unambiguous* if $\mathrm{da}_A^{\sup} = 1$. Clearly, every deterministic automaton is unambiguous.

3 Problems on Tree Width and Ambiguity for NPDAs

Okhotin and Salomaa [16] show that deciding whether a given NPDA has tree width (at most) k is undecidable for $k \geq 3$. Additionally, it is shown in [16] that for a given $k \in \mathbb{N}$ one can decide in polynomial time whether or not the tree width of a given NIDPDA is k. However, the question whether we can decide whether the tree width of a given NIDPDA is finite is left open in [16].

We give an algorithm that decides finiteness of tree width in polynomial time even for general NPDAs. The crucial observation is that, in fact, the algorithm does not need to rely on specific properties of input-driven computation and, instead, it is sufficient to check for an NPDA M whether some nondeterministic transition can be used an unbounded number of times, that is, the tree width of M is infinite if and only if in computations of M some nondeterministic transition can be used an unbounded number of times.

The above property is stated formally in Lemma 1 and the proof of the lemma follows directly from the definition of computation trees and tree width using the fact that the number of transitions is finite.

For λ-transitions we have to be a little more precise in defining the use of a nondeterministic transition. In the below lemma when saying that computation on input w *uses a nondeterministic transition* $(q, \lambda, X) \to (p, u)$ this means that either (i) the λ-transition is inherently nondeterministic, or (ii) the λ-transition is Z-nondeterministic and the next input character belongs to $Z \subseteq \Sigma$.

Lemma 1. *For an NPDA M, $\mathrm{tw}_M^{\sup}$ is infinite if and only if there exists a nondeterministic transition $(q, b, X) \to (p, u)$ of M such that for all $c \in \mathbb{N}$ there exists a word w such that M has a computation on w that uses the transition $(q, b, X) \to (p, u)$ as a* nondeterministic transition[4] *more than c times.*

Theorem 2. *We can decide whether or not the tree width of a given NPDA is finite in polynomial time.*

Proof. Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be an NPDA. By Lemma 1, the tree width of P is infinite if and only if there exist computations that use an unbounded number nondeterministic steps involving some nondeterministic transition $(q, b, X) \to (p, u)$ where $q, p \in Q$, $b \in \Sigma \cup \{\lambda\}$, $X \in \Gamma$, $u \in \Gamma^*$. Consequently we can decide whether or not the tree width of P is infinite by finding such a transition. A technical issue we need to take care of is that a Z-nondeterministic ($Z \subseteq \Sigma$) λ-transition involves a nondeterministic step only if the next input character belongs to Z.

Consider $q, p \in Q$, $b \in \Sigma \cup \{\lambda\}$, $X \in \Gamma$ and $u \in \Gamma^*$ such that the transition $t = (q, b, X) \to (p, u)$ is nondeterministic. When $b = \lambda$, this means that either $(q, \lambda, X) \to (p, u)$ is inherently nondeterministic or it is Z-nondeterministic for some $\emptyset \neq Z \subseteq \Sigma$. From the given NPDA P, for a nondeterministic transition t we construct a new NPDA P_t which, roughly speaking, reads a character only when simulating a nondeterministic step of P involving the transition t.

We have three cases to consider. In cases 1. and 2. the set of states of P_t will be the same as the set of states of the original NPDA P but in the third case P_t will need additional states.

1. When $t = (q, b, X) \to (p, u)$ is not a λ-transition, i.e., $b \in \Sigma$, we construct the NPDA P_t by replacing the input labels of all transitions except the transition t by λ and replacing the set of final states by the singleton set $\{q\}$.
2. If $b = \lambda$ and $t = (q, \lambda, X) \to (p, u)$ is inherently nondeterministic, P_t is constructed from P by replacing the input labels of all transitions other than t by λ and, furthermore, the transition t is replaced by $(q, \$, X) \to (p, u)$ where \$ is a new input symbol.
3. Suppose then that $t = (q, \lambda, X) \to (p, u)$ is Z-nondeterministic for $\emptyset \neq Z \subseteq \Sigma$, i.e., t is not inherently nondeterministic. If Q is the state set of P, the states of P_t will have a second component, i.e., state set of P_t is $Q \times \{0, 1\}$. The purpose of the second components is just to enforce that after applications of t that are "counted as a nondeterministic transition", the next real transition of P that is simulated must use an input symbol in the set Z.
   In real transitions $t' = (r, d, Y) \to (s, v)$, $d \in \Sigma$, the input symbols are again replaced by λ. More precisely, if $d \notin Z$, t' is replaced by $((r, 0), \lambda, Y) \to ((s, 0), v)$ and if $d \in Z$, t' is replaced by both $((r, 0), \lambda, Y) \to ((s, 0), v)$ and
   $$((r, 1), \lambda, Y) \to ((s, 0), v). \tag{1}$$

[4] This is explained in the paragraph before the lemma. Whether or not a λ-transition involves a nondeterministic step may depend on the next input symbol.

λ-transitions $t' = (r, \lambda, Y) \to (s, v)$ that are distinct from t are replaced by $((r, 0), \lambda, X) \to ((s, 0), v)$ and $((r, 1), \lambda, X) \to ((s, 1), v)$. Finally, the transition t itself is replaced by $((q, 0), \$, X) \to ((p, 1), u)$, $((q, 1), \$, X) \to ((p, 1), u)$, and $((q, 0), \lambda, X) \to ((p, 0), u)$.

In cases 1. and 2. it is clear that P_t accepts an infinite number of input strings if and only if P has computations that use the transition t an unbounded number of times. Note that in case 1. and 2. all applications of the transition t are nondeterministic.

The construction in the case 3. enforces that P_t accepts an infinite number of strings if and only if P has computations that use t *as a nondeterministic transition* an unbounded number of times. This is verified as follows. The only real (non-λ) transitions of P_t are

$$((q, 0), \$, X) \to ((p, 1), u) \text{ and } ((q, 1), \$, X) \to ((p, 1), u).$$

Thus, applying a real transition of P_t changes the second component of the state to 1 (or makes the second component remain 1 if it was 1 already). When the second component of the state is one, only transitions that simulate a λ-transition of P can be applied until we apply a transition (1) that simulates a real transition of P on $d \in Z$, that is, on an input symbol with respect to which the λ-transition t is nondeterministic.

Our algorithm constructs the NPDA P_t for each nondeterministic transition t of P. Thus, we can decide whether the tree width of P is infinite by checking whether for some nondeterministic transition t the language $L(P_t)$ is infinite. Otherwise, our algorithm answers that the tree width of P is finite. We describe the process in pseudocode in Algorithm 1.

Algorithm 1: Algorithm deciding the finiteness of the tree width of P

```
Input  : An NPDA P = (Q, Σ, Γ, δ, q0, Z0, F)
Output : Finiteness of the tree width of P
foreach transition t of P do
    if t is nondeterministic then
        Construct Pt from P;
        if |L(Pt)| = ∞ then
            return False
        end
    end
end
return True
```

Clearly, our algorithm terminates in polynomial time since testing the finiteness of a context-free language can be done in polynomial time. First we convert an NPDA into a context-free grammar using the standard triple construction [10] and again convert the grammar into the Chomsky normal form. Then, we draw a

dependency graph with nodes labeled by nonterminals of the context-free grammar. For instance, if there is a rule such as $S \to AB|BC|a$, we draw edges from S to A, B and C. Now we can decide whether or not the context-free grammar generates an infinite number of strings by finding any cycle from the nonterminal dependency graph. Since we repeat the above procedure $|Q|^2 \cdot (|\Sigma|+1) \cdot |\Gamma|^3$ times in the worst case, the total procedure terminates in polynomial time. Strictly speaking, instead of all transitions, it is sufficient to enumerate the left sides of the transitions and the above estimation can obviously be improved. □

Thus, for a given NPDA we can decide in polynomial time whether the tree width is finite and deciding whether the tree width equals $k \geq 3$ is undecidable [16]. Next we consider the remaining case of k being 1 or 2.

Proposition 3. *For a given NPDA P and an integer $k \in \{1, 2\}$, it is decidable whether or not the tree width of P is bounded by k in polynomial time.*

Proof. Consider the case when $k = 1$. Given an NPDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ and a transition of P, we can check whether the transition is useful by changing the target state of the transition to the only final state of P and checking whether the modified NPDA accepts any string. Since the emptiness of an NPDA can be decided in polynomial time, we can find useful transitions in polynomial time. Then, we check whether or not there exists a nondeterministic transition which is useful in P. If there is such a transition, we can decide that the tree width of P is not bounded by 1 because we can reach the state and choose a transition between two choices.

Let us consider the case when $k = 2$. Similarly to the previous case, we first find useful transitions from P. If there exists a useful transition with at least 3 nondeterministic choices, then the tree width is at least 3. Otherwise, we need to check whether a nondeterministic transition is reachable from another nondeterministic transition.

First we choose a nondeterministic useful transition of P that corresponds to a tuple (q, a, A). We modify P by replacing the labels of all transitions by λ except the chosen transition and change the target state, say p, of the chosen transition to p' which is the created copy of p. Let us change the label of the chosen transition to a special symbol $\star$, where $\star \notin \Sigma \cup \{\lambda\}$.

So far, the only transition with a non-empty label is the chosen transition. Then, we choose a nondeterministic useful transition from the copied NPDA and change the target state of the transition to the only final state. It is easy to verify that the only string that can be accepted by the constructed NPDA is $\star$ after two nondeterministic choices. Since there are a polynomial number of pairs of nondeterministic transitions of P and the acceptance of the string $\star$ can be decided in polynomial time, we can decide whether or not the tree width of P is bounded by 2 in polynomial time. □

Harrison [9] shows using a reduction from the Post Correspondence Problem that it is undecidable whether a context-free grammar G is ambiguous and a simple modification of the proof shows that finite ambiguity is undecidable for

context-free grammars.[5] By converting the grammars to pushdown automata it follows that the same property is undecidable for NPDAs.

For the sake of completeness, we show below that for an NPDA P it is undecidable whether the ambiguity of P is bounded by a fixed $k \in \mathbb{N}$. The proof is modified from a construction showing that deciding the exact tree width of an IDPDA is undecidable [16].

Let $\Sigma = \{a, b\}$. Recall that an instance of the Post Correspondence Problem (PCP) is a set of pairs of words $I = \{(u_1, v_1), \ldots, (u_n, v_n)\} \subseteq \Sigma^* \times \Sigma^*$, $n \geq 1$, and a solution to the instance I a sequence of indices $\ell_1, \ldots, \ell_k$ with each $1 \leq \ell_i \leq n$ such that: $u_{\ell_1} \cdots u_{\ell_k} = v_{\ell_1} \cdots v_{\ell_k}$. It is well known that for a given PCP instance I it is undecidable to determine whether or not I has a solution [9, 10, 19, 21].

Proposition 4. *Let $k \geq 1$ be fixed. For a given NPDA P it is undecidable whether or not the degree of ambiguity of P is bounded by k.*

4 Problems on Tree Width for NIDPDAs and NFAs

It is known that we can decide whether or not the tree width of an NIDPDA is bounded by a fixed integer k in polynomial time [16]. However, for the complexity of deciding whether tree width is bounded by a variable k, only an EXPTIME upper bound has been known.

Theorem 5. *Given an NIDPDA A and an integer $k \geq 1$, the problem of determining whether or not the tree width of A is bounded by k is* EXPTIME*-complete.*

Proof. Note that the EXPTIME upper bound is already proven by Okhotin and Salomaa [16]. They have shown that for an NIDPDA A with n states and an integer $k \geq 0$, we can decide whether or not $\mathrm{tw}_A^{\mathrm{sup}} \leq k$ in time $poly(k^k \cdot n^k)$.

We prove the EXPTIME-hardness by reduction from the intersection emptiness of deterministic top-down tree automata [6]. It is known that a deterministic top-down tree automaton can be translated into a DIDPDA in polynomial time [2]. Thus, we can convert k deterministic top-down tree automata into k DIDPDAs in polynomial time and then, the intersection emptiness of the k DIDPDAs is also EXPTIME-hard (in fact, EXPTIME-complete).

Now we construct a new NIDPDA A that has the initial state connected to the initial states of k DIDPDAs by transitions on a new local symbol \$. We first create a new final state f of A which will be the only final state of A. This implies that we change all final states of the k DIDPDAs into non-final states. Then, for each final state of all k DIDPDAs, we create two nondeterministic transitions labelled by a new symbol '#' that go to the sole final state f of A. It is easy to see that the tree width of A is $2k$ if and only if the k DIDPDAs have a non-empty intersection.

We can see that the problem of deciding whether or not the tree width of an NIDPDA is bounded by a given integer $2k$ decides also whether the intersection of k deterministic top-down tree automata is empty. As a result, we conclude that the k-tree width problem for NIDPDAs is EXPTIME-complete. □

[5] For original references see Wood [21].

Now we consider the tree width of an NFA. It is known that there is a simple polynomial time algorithm to decide whether the tree width of an NFA is finite [17]. Here we show that the problem of deciding whether or not the tree width of an NFA is bounded by a given integer k is PSPACE-complete. The proof is inspired by the corresponding result for the degree of ambiguity by Chan and Ibarra [5].

Theorem 6. *Given an NFA A and an integer $k \geq 1$, the problem of determining whether or not the tree width of A is bounded by k is* PSPACE*-complete.*

Proof. Let $A = (\Sigma, Q, q_0, F, \delta)$ be an input NFA. We construct a DFA $A' = (\Sigma, Q', q_0', F', \delta')$, where $Q' = (Q \cup \#)^{k+1}$ is the set of states, $q_0' = (q_0, \#, \ldots, \#)$ is the initial state, $F' = \{(q_1, q_2, \ldots, q_{k+1}) \in Q^{k+1} \mid |\{q_1, q_2, \ldots, q_{k+1}\} \cap Q| = k+1\}$ is the set of final states, and δ' is the transition function that simulates at most $k+1$ computations of A in the $(k+1)$-tuple states in Q' simultaneously. From the initial state $q_0' = (q_0, \#, \ldots, \#)$, A' changes its first component according to the transition function δ of A. For instance, if $\delta(q_0, a) = \{p\}$, then A' moves from $(q_0, \#, \ldots, \#)$ to $(p, \#, \ldots, \#)$ by reading the character a. If A' encounters a nondeterministic computation step such as $\delta(p, a) = \{p_1, p_2\}$ from $(p, \#, \ldots, \#)$, then A' moves to $(p_1, p_2, \#, \ldots, \#) \in Q'$ by reading a. The idea is to replace the first '#' entries of the $(k+1)$-tuple into the target states of the nondeterministic choices of A. In this way, we can decide whether or not the tree width of A is k since $L(A')$ is empty if and only if the tree width of A is k.

Obviously, the number of states in A' is bounded by $(n+1)^{k+1}$. The nonemptiness of $L(A')$ can be decided in nondeterministic polynomial space by guessing a candidate string w of length smaller than $(n+1)^{k+1}$.

The PSPACE-hardness can be obtained by slightly modifying the proof of Theorem 5 on the EXPTIME-hardness reduction. If we simply replace the k DIDPDAs with k DFAs, then we can see that the problem of deciding whether or not the tree width of an NFA is k is PSPACE-hard from the PSPACE-completeness of the DFA intersection emptiness [12]. □

5 Conclusion

As the main result we have shown that the question whether the tree width of an NPDA is finite can be decided in polynomial time. This question was previously open even for NIDPDAs. The main open problem is whether or not for input-driven pushdown automata with unbounded ambiguity, the degree of growth of ambiguity can be decided effectively. Similarly questions can naturally be asked also about tree width growth rates. The elegant structural characterization given by Weber and Seidl [20] for NFAs with polynomial or exponential ambiguity growth rates does not seem to work in the presence of stack operations. On the other hand, when stack operations are input-driven, showing undecidability using a reduction from the Post Correspondence Problem, analogously as done e.g. in Proposition 4, clearly is not possible either.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing. pp. 202–211 (2004)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM **56**(3), 16:1–16:43 (2009)
3. Björklund, H., Martens, W.: The tractability frontier for NFA minimization. Journal of Computer and System Sciences **78**(1), 198–210 (2012)
4. Caralp, M., Reynier, P.A., Talbot, J.M.: Visibly pushdown automata with multiplicities: Finiteness and k-boundedness. In: Proceedings of DLT'12. pp. 226–238 (2012)
5. hung Chan, T., Ibarra, O.H.: On the finite-valuedness problem for sequential machines. Theoretical Computer Science **23**(1), 95–101 (1988)
6. Fernau, H., Krebs, A.: Problems on finite automata and the exponential time hypothesis. Algorithms **10**(1) (2017)
7. Goldstine, J., Kappes, M., Kintala, C.M., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. Journal of Universal Computer Science **8**(2), 193–234 (2002)
8. Goldstine, J., Leung, H., Wotschke, D.: Measuring nondeterminism in pushdown automata. Journal of Computer and System Sciences **71**(4), 440–466 (2005)
9. Harrison, M.: Introduction to Formal Language Theory. Addison-Wesley, Reading, MA (1978)
10. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA, 2 edn. (1979)
11. Hromkovič, J., Seibert, S., Karhumäki, J., Klauck, H., Schnitger, G.: Communication complexity method for measuring nondeterminism in finite automata. Information and Computation **172**(2), 202–217 (2002)
12. Kozen, D.: Lower bounds for natural proof systems. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science. pp. 254–266 (1977)
13. Leung, H.: Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. SIAM Journal on Computing **27**(4), 1073–1082 (1998)
14. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: Proceedings of the 7th International Colloquium on Automata, Languages, and Programming, pp. 422–435 (1980)
15. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News **45**(2), 47–67 (2014)
16. Okhotin, A., Salomaa, K.: Input-driven pushdown automata with limited nondeterminism. In: Proceedings of DLT'14. pp. 84–102 (2014)
17. Palioudakis, A., Salomaa, K., Akl, S.G.: State complexity of finite tree width NFAs. Journal of Automata, Languages and Combinatorics **17**(2), 245–264 (2012)
18. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. 3: Beyond Words. Springer-Verlag New York, Inc. (1997)
19. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2009)
20. Weber, A., Seidl, H.: On the degree of ambiguity of finite automata. Theoretical Computer Science **88**(2), 325–349 (1991)
21. Wood, D.: Theory of Computation. Harper & Row (1986)