The No Endmarker Theorem for One-Way Probabilistic Pushdown Automata

Tomoyuki Yamakami

Abstract

In various models of one-way pushdown automata, the explicit use of two designated endmarkers on a read-once input tape has proven to be extremely useful for making a conscious, final decision on the acceptance/rejection of each input word right after reading the right endmarker. With no endmarkers, by contrast, a machine must constantly stay in either accepting or rejecting states at any moment since it never notices the end of the input instance. This situation, however, helps us analyze the behavior of the machine whose tape head makes the consecutive moves on all prefixes of a given extremely long input word. Since those two machine formulations have their own advantages, it is natural to ask whether the endmarkers are truly necessary to correctly recognize languages. In the deterministic and nondeterministic models, it is well-known that the endmarkers are removable without changing the acceptance criteria of each input instance. This paper proves that, for a more general model of one-way probabilistic pushdown automata, the endmarkers are also removable. This is proven by employing probabilistic transformations from an “endmarker” machine to an equivalent “no-endmarker” machine at the cost of double exponential state complexity without compromising its error probability. By setting this error probability appropriately, our proof also provides an alternative proof to both the deterministic and the nondeterministic models as well.

Keywords. probabilistic pushdown automata, endmarker, descriptional complexity, acceptance criteria, stack-state complexity

1 With or Without Endmarkers, That is a Question

1.1 Two Different Formulations of One-Way Pushdown Automata

In automata theory, pushdown automata are regarded as one of the most fundamental architectures operated with finite-state controls. A pushdown automaton is, in general, a finite-controlled machine equipped with a special memory device, called a stack, in which information is stored and accessed in the first-in last-out manner. Here, we are focused on one-way pushdown automata whose tape heads either move to the right or stay still at any step. In particular, a nondeterministic variant of those machines, known as one-way nondeterministic pushdown automata (or 1npda’s), characterizes context-free languages. Similarly, one-way deterministic pushdown automata (or 1dpda’s) introduce deterministic context-free languages. These machines can be seen as special cases of a much general model of one-way probabilistic pushdown automata (or 1ppda’s). Macarie and Ogihara [8] discussed the computational complexity of languages recognized by 1ppda’s with unbounded-error probability. In recent literature, Hromkovič and Schnitger [4] and Yamakami [15] further discussed the limitations of the power of bounded-error 1ppda’s.

In textbooks and scientific papers, ignoring small variational deviations, various one-way pushdown automata have two distinct but widely-used formulations. At the first glance, those formulations look quite different and it is not obvious that they are essentially “equivalent” in recognition power. In the first formalism, an input string over a fixed alphabet is initially given to an input tape together with two designated endmarkers, which mark both ends of the input string. The machine starts at scanning the left endmarker $\text{c}$ and moves its tape head to the right whenever it accesses an input symbol until it reaches the right endmarker $. In addition, the machine can proceed without reading any input symbol even after reading $. Such special steps are called $\lambda$-moves or a $\lambda$-transitions. Even if the machine halts even before reading the endmarker by entering halting states, the use of $\lambda$-moves makes it possible to postpone the time to halt until the machine reads the right endmarker. Whenever the machine enters a halting state (i.e., either an accepting state or a rejecting state), the machine is considered to halt and its computation terminates in acceptance or rejection depending on the type of halting states.

In the second formalism, by sharp contrast, the input tape contains only a given input string without the presence of two endmarkers and the machine starts reading the input from left to right until it reads off the rightmost symbol. Since the machine need to access the entire input string without knowing the end of the input, whenever the machine reads off the input, it is considered to “halt” and the current inner state of the machine determines the acceptance and rejection of the input.
For convenience, we call a model described by the first formalism an endmarker model and one by the second formalism a no-endmarker model. Kajeps, Geidmanis, and Freivalds [6], Hromković and Schnitger [4], and Yamakami [15] all used a no-endmarker model of 1ppda’s (succtingly called no-endmarker 1ppda’s) in their analyses of machine’s behaviors, whereas an endmarker model of 1ppda’s (called endmarker 1ppda’s) was used by Macarie and Ogihara [8] and Yamakami [16]. It is commonly assumed that endmarker 1ppda’s are equivalent in recognition power to no-endmarker 1ppda’s. Unfortunately, there is no “formal” proof for the equivalence between those two models without compromising error probability.

The roles of two endmarkers, the left and the right endmarkers, are clear. In the presence of the right endmarker $\$, in particular, when the machine reads $\$, the machine certainly notices the end of the input string and, based on this knowledge, it can make the final transition (followed by a possible series of $\lambda$-moves) before entering either accepting or rejecting states. In the presence of $\$, we can make a series of $\lambda$-moves to empty the stack before halting. Without the right endmarker, however, the machine must be always in a “constantly halting” inner state (either an accepting state or a rejecting state). Such an inner state determines the acceptance or rejection of an input just after reading off the entire input string and possibly following a series of $\lambda$-moves.

This halting condition certainly helps us analyze the behavior of the machine simply by tracing down the changes of accepting and rejecting states on all prefixes of any extremely long input string. For instance, some of the results of Hromković and Schnitger [4] and Yamakami [15] were proven in this way.

For 1dpda’s as well as 1npda’s, Hopcroft and Ullman [5] used in their textbook the no-endmarker model to define pushdown automata whereas Lewis and Papadimitriou [7] suggested in their textbook the use of the endmarker model. For those basic pushdown automata, the endmarker model and the no-endmarker model are in fact well-known to be “equivalent” in their computational power. We succinctly call this assertion the no endmarker theorem throughout this paper.

For 1npda’s, for instance, we can easily convert one model to the other without compromising its acceptance/rejection criteria by first transforming a 1npda $M$ to its equivalent context-free grammar, converting it to Greibach Normal Form, and then translating it back to its equivalent 1npda (see, e.g., [3]).

The question of whether the endmarker models of one-way pushdown automata can be computationally equivalent to the no endmarker models of the same machine types is so fundamental and also useful in the study of various types of one-way pushdown automata. This paper extends our attention to 1ppda’s—a probabilistic variant of pushdown automata. For such 1ppda’s, we wish to argue whether the no endmarker theorem is indeed true because, unfortunately, not all types of pushdown automata enjoy the no endmarker theorem. The right endmarker can be eliminated if our machine model is closed under right quotient with regular languages (see, e.g., [3]). Since 1dpda’s and 1npda’s satisfy such a closure property [2], the no endmarker theorem holds for those machine models. For counter machines (i.e., pushdown automata with single letter stack alphabets except for the bottom marker), on the contrary, the endmarkers are generally unremovable [4]; more precisely, the deterministic reverse-bounded multi-counter machines with endmarkers are generally more powerful than the same machines with no endmarkers. This latter fact demonstrates the crucial role of the endmarkers for pushdown automata. It is therefore significantly important to discuss the removability of the endmarkers for bounded-error and unbounded-error 1ppda’s.

### 1.2 The No Endmarker Theorem

This paper presents the proof of the no endmarker theorem for 1ppda’s with arbitrary error probability. More precisely, we prove the following statement, which allows us to safely remove the two endmarkers of 1ppda’s without compromising error probability.

**Theorem 1.1** [No Endmarker Theorem] Let $\Sigma$ be any alphabet and let $\varepsilon : \Sigma^* \rightarrow [0, 1/2)$ be any error-bound parameter (when $\varepsilon(x)$ is a one-sided error, we can take $\varepsilon(x) \in (0, 1)$ instead). For any language $L$ over $\Sigma$, the following two statements are logically equivalent.

1. There exists a 1ppda with two endmarkers that recognizes $L$ with error probability $\varepsilon(x)$ on every input $x$.
2. There exists a 1ppda with no endmarker that recognizes $L$ with error probability $\varepsilon(x)$ on every input $x$.

The first part of Theorem 1.1 for 1ppda’s asserts that we can safely eliminate the two endmarkers from each 1ppda without changing the original error probability. The invariance of this error probability is important because this invariance makes it possible to apply the same proof of ours to both 1dpda’s and 1npda’s as special cases by setting $\varepsilon(x) = 0$ for all $x \in \Sigma^*$ and by making $\varepsilon(x) = 0$ for all $x \in L$ and $\varepsilon(x) < 1$ for all $x \in L$, respectively. Thus, we obtain the well-known fact that the endmarkers are removable for 1dpda’s and 1npda’s.
Corollary 1.2 Theorem 1.1 also holds for 1dpda’s as well as 1npda’s.

Since the proof of Theorem 1.1 is constructive, it is possible to discuss the increase of the number of inner states and of stack alphabet size in the construction of new machines. Another important factor is the maximal size of stack strings stored by “push” operations. We call this number the push size of the machine. Throughout this paper, altogether of those three factors are succinctly referred to as the stack-state complexity of transforming one model to the other.

For brevity, we say that two 1ppda’s (with or without endmarkers) are error-equivalent if their outputs agree with each other on all inputs with exactly the same error probability.

Proposition 1.3 Given an n-state no-endmarker 1ppda with stack alphabet size m and push size e, there is an error-equivalent endmarker 1ppda of stack alphabet size m + 2 and push size e with at most 2n + 1 states.

Proposition 1.4 Given an n-state endmarker 1ppda with arbitrary push size, there is an error-equivalent no-endmarker 1ppda such that it is double-exponential in state size and stack alphabet size but is of push size 1.

Since Theorem 1.1 follows directly from Propositions 1.3–1.4, we will concentrate our efforts on proving these propositions in the rest of this paper. From the propositions, we can observe that endmarker 1ppda’s are likely to be double-exponentially more succinct in descriptional complexity than no-endmarker 1ppda’s. We are not sure, however, that this bound can be significantly improved.

Organization of This Paper. In Section 2 we will start with the formal definition of 1ppda’s (and their variants, 1dpda’s and 1npda’s) with or without endmarkers. In Section 3 we will prove Proposition 1.3. Proposition 1.4 will be proven in Section 5. To simplify the proof of Proposition 1.4 in Section 4, we will transform each standard 1ppda into another special 1ppda that does not halt before the right endmarker and takes a “push-pop-controlled” form, called an ideal shape, which turns out to be quite useful in proving various properties of languages. As a concrete example of usefulness, we will demonstrate in Section 4.3 the closure property of language families induced by bounded-error endmarker 1ppda’s under “reversal”. Lastly, a few open questions regarding the subjects of this paper will be discussed in Section 6.

2 Various One-Way Pushdown Automata

Let us review two machine models of 1ppda’s in details, focusing on the use of the two endmarkers. A one-way probabilistic pushdown automaton (abbreviated as 1ppda) M runs essentially in a way similar to a one-way nondeterministic pushdown automaton (or a 1npda) except that, instead of making a nondeterministic choice at each step forming a number of computation paths, M randomly chooses one of all possible transitions and then branches out to produce multiple computation paths. The probability of such a computation path is determined by a series of random choices made along the computation path. The past literature has taken two different formulations of 1ppda’s, with or without two endmarkers. We will explain these two formulations in the subsequent subsections.

2.1 Numbers, Strings, and Languages

Let N denote the set of all natural numbers, including 0, and set N to be N - {0}. Given a number n ∈ N, [n] denotes the set {1, 2, ..., n}. Given a set A, P(A) denotes the power set of A; namely, the set of all subsets of A.

An alphabet is a finite nonempty set of “letters” or “symbols.” Given an alphabet Σ, a string over Σ is a finite sequence of symbols in Σ. Conventionally, we use λ to express the empty string as well as the pop operand for a stack. The length of a string x, expressed as |x|, is the total number of symbols in x. For a string x = x_1x_2⋯x_{n−1}x_n over Σ with x_i ∈ Σ for any index i ∈ [n], the reverse of x is the string x_nx_{n−1}⋯x_2x_1 and is denoted by x^R.

The notation Σ* indicates the set of all strings over Σ whereas Σ+ expresses Σ* - {λ}. A language over Σ is a subset of Σ*. For two languages A and B, AB denotes the concatenation of A and B, that is, {xy | x ∈ A, y ∈ B}. In particular, when A = {w}, we write wB instead of {w}B. Similarly, when B = {w}, we use the notation Aw. For a number k ∈ N, A^k expresses the set {x_1x_2⋯x_k | x_1, x_2, ..., x_k ∈ A}. Moreover, we use the notation [A, B] for two sets A and B to denote the set {[a, b] | a ∈ A, b ∈ B} of “bracketed” ordered pairs.
2.2 The First Formulation

We start with explaining the first formulation of 1ppda’s whose input tapes marked by two designated endmarkers. We always assume that those endmarkers are not included in any input alphabet. As discussed in, e.g., [7], a 1ppda with two endmarkers (which we call an endmarker 1ppda) has a read-once semi-infinite input tape, on which an input string is initially placed, surrounded by two endmarkers $\langle$ (left endmarker) and $\rangle$ (right endmarker). To clarify the use of those two endmarkers, we explicitly include them in the description of a machine as $(Q, \Sigma, \{\langle, \rangle\}, \Gamma, \Theta, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$, where $Q$ is a finite set of inner states, $\Sigma$ is an input alphabet, $\Gamma$ is a stack alphabet, $\Theta$ is a finite subset of $\Gamma^*$ with $\lambda \in \Theta$, $\delta : ((Q - Q_{halt}) \times \Sigma \times \Gamma \times \Theta \times \Gamma \rightarrow [0, 1]$ (with $\Sigma = \Sigma \cup \{\langle, \rangle\}$ and $\Sigma_{\lambda} = \Sigma \cup \{\lambda\}$) is a probabilistic transition function, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the bottom marker, $Q_{acc} \subseteq Q$ is a set of accepting states, and $Q_{rej} \subseteq Q$ is a set of rejecting states, where $Q_{halt}$ denotes $Q_{acc} \cup Q_{rej}$. We always assume that $Q_{halt} \cap Q_{rej} = \emptyset$. For convenience, let $\Gamma_{\lambda} = \Gamma - \{Z_0\}$. The minimum positive integer $e$ for which $\Theta \subseteq \Gamma_{\lambda}^e$ is called the push size of $M$. Each value $\delta(q, \sigma, a, p, u)$ expresses the probability that, when $M$ reads $\sigma$ on the input tape and $a$ in the top of the stack in inner state $q$, $M$ changes $q$ to another inner state $p$, and replaces $a$ by $u$.

We remark that, in certain literature, a pushdown automaton is assumed to halt just after reading $\langle$ without making any extra $\lambda$-move. For a general treatment of 1ppda’s, this paper allows the machine to make a (possible) series of $\lambda$-moves even after reading $\langle$ until it finally enters a halting state. After reading $\langle$, the tape head is considered to move off (or leave) the input region, which is marked as $\langle: \rangle$ on the input tape.

To deal with 1npda’s $\lambda$-moves, for any segment $x$ of input $\psi w \langle$, we say that $x$ is completely read if $M$ reads all symbols in $x$, moves its tape head off the string $x$, and makes all (possible) $\lambda$-moves after reading the last symbol of $x$. At each step, $M$ probabilistically selects either a $\lambda$-move or a non-$\lambda$-move, or both. Conveniently, we define $\delta(q, \sigma, a | p, u) = \sum_{(p, u) \in Q \times \epsilon_{\Gamma}} \delta(q, \sigma, a | p, u)$ for any triplet $(q, \sigma, a) \in Q \times \Sigma \times \Gamma$. We demand that $\delta$ satisfies the following probability requirement: $\delta(q, \sigma, a | p, u)$ is the probability that, when $M$ reads $\sigma$ on the input tape and $a$ in the top of the stack in inner state $q$, $M$ changes $q$ to another inner state $p$, and replaces $a$ by $u$.

To describe the behaviors of a stack, we follow [13] for the basic terminology. A stack content means a series $z = z_mz_{m-1} \cdots z_1z_0$ of stack symbols in $\Gamma$, which are stored inside the stack sequentially from the topmost symbol $z_m$ of the stack to the lowest symbol $z_0 (\neq Z_0)$. The bottom marker $Z_0$ is not popped, we often say that the stack is empty if there is no symbol in the stack except for $Z_0$.

A (surface) configuration of $M$ is a triplet $(q, i, w)$, which indicates that $M$ is in inner state $q$, $M$’s tape head scans the $i$th cell, and $M$’s stack contains $w$. The initial configuration of $M$ is $(q_0, 0, Z_0)$. An accepting (resp., a rejecting) configuration is a configuration with an accepting (resp., a rejecting) state and a halting configuration is either an accepting or a rejecting configuration. Given a fixed input $x \in \Sigma^*$, we say that a configuration $(p, j, uw)$ follows another configuration $(q, i, w)$ with probability $\delta(q, \sigma, a | p, u)$ if $i$ is the $\sigma$th symbol of $x$, and $j = i$ if $\sigma = \lambda$ and $j = i + 1$ if $\sigma \neq \lambda$.

A computation path of length $k$ is a series of $k$ configurations, which describes a history of consecutive “moves” chosen by $M$ on input $x$, starting at the initial configuration with probability $p_0 = 1$ and, for each index $i \in [0, k - 1]$, the $i + 1$st configuration follows the $i$th configuration with probability $p_i$, ending at a halting configuration with probability $p_k$. To such a computation path, we assign the probability $\Pi_{i \in [0, k]} p_i$. A computation path is called accepting (rejecting, resp.) if the path ends with an accepting configuration (a rejecting configuration, resp.). Generally, a 1ppda may produce an extremely long computation path or even an infinite computation path; therefore, we must restrict our attention to finite computation paths.

Hromkovič and Schnitger [4] and Kuipers, Geidemanis, and Freivalds [6] both used a model of 1ppda’s whose computation paths all halt eventually (i.e., in finitely many steps) on every input. We also adopt this convention in the rest of this paper. In what follows, we always assume that all 1ppda’s should satisfy this requirement. Standard definitions of 1dpda’s and 1npda’s do not have such a runtime bound, because we can easily convert those machines to ones that halt within $O(n)$ time (e.g., [3, 12, 14]).
The acceptance probability of $M$ on input $x$ is the sum of all probabilities of accepting computation paths of $M$ starting with $\epsilon x\$ on its input tape. We express this by $p_{M,\text{acc}}(x)$ the acceptance probability of $M$ on $x$. Similarly, we define $p_{M,\text{rej}}(x)$ to be the rejection probability of $M$ on $x$. Whenever $M$ is clear from the context, we often omit script “M” entirely and write, e.g., $p_{\text{acc}}$ instead of $p_{M,\text{acc}}$. We further say that $M$ accepts (resp., rejects) $x$ if the acceptance (resp., rejection) probability $p_{M,\text{acc}}(x)$ (resp., $p_{M,\text{rej}}(x)$) is more than 1/2 (resp., at least 1/2). Since all computation paths are assumed to halt in linear time, for any given string $x$, either $M$ accepts it or $M$ rejects it. The notation $L(M)$ stands for the set of all strings $x$ accepted by $M$; that is, $L(M) = \{ x \in \Sigma^* \mid p_{M,\text{acc}}(x) > 1/2 \}$.

Given a language $L$, we say that $M$ recognizes $L$ if $L$ coincides with the set $L(M)$. A 1ppda $M$ is said to make bounded error if there exists a constant $\varepsilon \in [0,1/2]$ (called an error bound) such that, for every input $x$, either $p_{M,\text{acc}}(x) \geq 1 - \varepsilon$ or $p_{M,\text{rej}}(x) \geq 1 - \varepsilon$; otherwise, we say that $M$ makes unbounded error.

Regarding the behavioral equivalence of probabilistic machines, two 1ppda’s $M_1$ and $M_2$ are said to be error-equivalent to each other if $L(M_1) = L(M_2)$ and $M_1$ simulates $M_2$ (or $M_2$ simulates $M_1$) with exactly the same error probability on every input. In this paper, we are also concerned with the descriptive complexity of model machines. We use the following three complexity measures to describe each machine $M$. The state size (or state complexity) of $M$ is $|Q|$, the stack alphabet size of $M$ is $|\Gamma|$, and the push size of $M$ is the maximum length of any string in $\Theta$. It then follows that $|\Theta| \leq m^c$ if $M$ has stack alphabet size $m$ and push size $c$.

### 2.3 The Second Formulation

In comparison with the first formulation, let us consider 1ppda’s with no endmarker. Such machines are succinctly called no-endmarker 1ppda’s and they are naturally obtained from all the definitions stated in Section 2.2 except for removing the entire use of $\xi$ and $\$$ for example, an input region is now the cells that contain $x$, instead of $\epsilon x\$. We express such a machine as $((Q, \Sigma, \Theta, \delta, q_0, Z_0, Q_{\text{acc}}, Q_{\text{rej}})$ without $\xi$ and $\$$ for such a no-endmarker 1ppda, its probabilistic transition function $\delta$ maps $Q \times \Sigma \times \Gamma \times Q \times \Theta$ to $[0,1]$, where $\Sigma = \Sigma \cup \{\lambda\}$. The acceptance and rejection of such no-endmarker 1ppda’s are determined by whether the 1ppda’s are respectively in accepting states or in rejecting states just after reading off the entire input string and leaving the input region. To ensure this, $Q$ must be partitioned into $Q_{\text{acc}}$ and $Q_{\text{rej}}$.

It is important to note that every 1ppda in the initial state must read the leftmost symbol of a given non-empty input string written on the input tape at the first step. In particular, when an input is the empty string, for any machine that has the right endmarker but no left endmarker, it can still read $\$ at the first step. In contrast, since a no-endmarker machine cannot “read” any blank symbol, we must allow the machine to make a $\lambda$-move at the first step.

### 2.4 Deterministic and Nondeterministic Variants

Deterministic and nondeterministic variants of 1ppda’s can be easily obtained by slightly modifying the definition of the 1ppda’s. To obtain a one-way deterministic pushdown automaton (or a 1dpda), we require $\delta(q, \sigma, a | p, u) \in \{0,1\}$ (instead of the unit real interval $[0,1]$) for all tuples $(q, \sigma, a, p, u)$. Similarly, we can obtain one-way nondeterministic pushdown automata (or 1npda’s) if we pick a possible next move uniformly at random and take the following acceptance/rejection criteria: a 1npda $M$ accepts input $x$ if $p_{M,\text{acc}}(x) > 0$ and $M$ rejects $x$ if $p_{M,\text{rej}}(x) = 1$. It is not difficult to see that these “probabilistic” definitions of 1dpda’s and 1npda’s coincide with the “standard” definitions written in many textbooks.

### 3 From No-Endmarker 1ppda’s to Endmarker 1ppda’s

Proposition[13] asserts a linear increase of the stack-state complexity of transforming no-endmarker 1ppda’s into error-equivalent endmarker 1ppda’s. Although the transformation seems rather easy, we briefly describe how to construct any no-endmarker 1ppda $M$ an error-equivalent endmarker 1ppda $N$.

**Proof of Proposition[13]**. Given any no-endmarker 1ppda $M = ((Q, \Sigma, \Gamma, \Theta, \delta, q_0, Z_0, Q_{\text{acc}}, Q_{\text{rej}})$, we construct an endmarker 1ppda $N = ((Q', \Sigma, \{\xi, \$$\}, \Gamma', \Theta', \delta', q_0', Z_0', Q'_{\text{acc}}, Q'_{\text{rej}})$, where $Q'_{\text{acc}} = \{ q \mid q \in Q_{\text{acc}} \}$, $Q'_{\text{rej}} = \{ q \mid q \in Q_{\text{rej}} \}$, $Q' = Q \cup \{ q_0' \} \cup Q'_{\text{acc}} \cup Q'_{\text{rej}}$, and $\Gamma' = \Gamma \cup \{ \xi, \$$\}$. It follows that $|Q'| \leq 2n + 1$ and $|\Gamma'| = m + 2$ and that the push size does not change.

Intuitively speaking, the behavior of $N$ is described as follows. In the first move, $N$ uses a new initial state $q_0'$ to read the left endmarker $\xi$ and then enters $M$’s initial state $q_0$ (in Line 1 below). Note that, while reading a non-$\lambda$ symbol, by the definition, $M$ does not halt even if it enters halting states. When $M$ is
in a halting state after reading the rightmost input symbol and making a series of (possible) λ-moves, N then reads $ and enters new but corresponding halting states without changing the stack content (in Line 3). Formally, we define the probabilistic transition function $\delta'$ as follows. Let $\Sigma_\lambda = \Sigma \cup \{\lambda\}$.

1. $\delta'(q_0, q, Z_0 | q_0, Z_0) = 1$
2. $\delta'(q, \sigma, a | p, w) = \delta(q, \sigma, a | p, w)$ for $q \in Q, a \in \Gamma$, and $\sigma \in \Sigma_\lambda$.
3. $\delta'(q, $, $ | a, a) = 1$ for $q \in Q_{\lambda\text{halt}}$ and $a \in \Gamma$ if $\delta[q, \lambda, a] = 0$.

It is not difficult to show that N simulates M correctly with the same error probability. This completes the proof of Proposition 3.3. \qed

4 Preparatory Machine Modifications

A key to our proof of Proposition 1.4 is the step to normalize the wild behaviors of endmarker 1ppda’s without compromising their error probability. This preparatory step is quite crucial and will help us prove the proposition significantly more easily in Section 5. In the nondeterministic model, it is always possible to eliminate all λ-moves of 1npda’s and limit the set $Q$ of inner states to $Q_{\text{simple}} = \{q_0, q, q_{\text{acc}}\}$ (this is a byproduct of translating a context-free grammar in Greibach Normal Form to a corresponding 1npda). For 1ppda’s, however, we can neither eliminate λ-moves nor limit $Q$ to $Q_{\text{simple}}$. Moreover, we cannot control the number of consecutive λ-moves. Despite all these difficulties, we can still curtail certain behaviors of 1ppda’s to control the execution of push and pop operations during their computations. Such a push-pop-controlled form is called “ideal shape.”

For 1npda’s and 1dpda’s, there are a few precursors in this direction: Hopcroft and Ullman 3, Chapter 10 and Pighizzini and Pisoni 11, Section 5. We intend to utilize the basic ideas of them and further expand the ideas to prove that all 1ppda’s can be transformed into their ideal-shape form.

4.1 No Halting Before Reading $$

We first modify a given endmarker 1ppda to another one with a certain nice property without compromising its error probability. In particular, we want to eliminate the possibility of premature halting well before reading $;$ namely, to force the machine to halt only on or after reading $;$ with a (possible) series of subsequent λ-moves. Recall that a stack is empty when it contains only the bottom marker.

Lemma 4.1 Given any endmarker 1ppda M with n states, m stack alphabet size, and e push size, there exists an error-equivalent endmarker 1ppda N having 2n + 2 states with stack alphabet size m and push size e such that it halts only on or after reading $;$; moreover, when N halts, its stack becomes empty.

A proof idea of Lemma 4.1 is to simulate M step by step until M either enters a halting state or reads $. When M enters a halting state before reading $,$ M remembers this state, empties the stack, continues reading input symbols, and finally enters a true halting state. On the contrary, when M reaches $,$ M remembers the passing of $,$ continues the simulation of M’s λ-moves. Once M enters a halting state, N remembers this state, empties the stack, and enters a true halting state.

For the clarity of the state complexity term of 2n + 2 in Lemma 4.1 and also for the later use of the lemma’s proof in Section 5.2 we include the detailed proof of the lemma.

Proof of Lemma 4.1 Let $M = (Q, \Sigma, \{\$\}, \Gamma, \Theta, \delta, q_0, Q_{\text{acc}}, Q_{\text{rej}})$ be any 1ppda with $|Q| = n, |\Gamma| = m$, and push size e. In the following, we define the desired 1ppda $N = (Q', \Sigma, \{\$\}, \Gamma', \Theta', \delta', q'_0, Q'_{\text{acc}}, Q'_{\text{rej}})$ that can simulate M with the same error probability. Firstly, we prepare a new non-halting inner state $\hat{q}$, which corresponds to each state q in $Q_{\text{halts}}$, and new accepting and rejecting states $q_{\text{acc}}'$ and $q_{\text{rej}}'$. We set $Q^{(0)} = \{q | q \in Q_{\text{rej}}\}$ and $Q^{(1)} = \{q | q \in Q_{\text{acc}}\}$. We also prepare $q^{(\$)}$ for each state q in $Q - Q_{\text{halts}}$ and define $Q^{(\$)} = \{q^{(\$)} | q \in Q\}$. Moreover, we define $Q_{\text{acc}} = \{q_{\text{acc}}'\}$, $Q'_{\text{rej}} = \{q'_{\text{rej}}\}$, $Q' = (Q - Q_{\text{halts}}) \cup Q^{(\$)} \cup Q^{(0)} \cup Q^{(1)} \cup Q_{\text{acc}} \cup Q_{\text{rej}}$, and $\Gamma' = \Gamma$. Recall from Section 2.2 the notation $\Sigma = \Sigma \cup \{\$\}$ and $\Sigma_\lambda = \Sigma \cup \{\lambda\}$.

In what follows, we describe how to define $\delta'$, which aims at simulating M step by step on a given input. Let us consider two cases, depending on whether or not M reads $$.\n
(1) When M enters an accepting state (resp., a rejecting state), say, q before reading $,$ N first enters its associated state $\hat{q}$, empties the stack, and continues reading the rest of an input string deterministically. When N finally reaches $ in state $\hat{q}$ in $Q_{\text{acc}}$ (resp., $Q_{\text{rej}}$), N changes it to $q_{\text{acc}}'$ (resp., $q_{\text{rej}}'$) and halt. Formally, $\delta'$ is set as follows. Let $q \in Q, p \in Q', a \in \Gamma, \sigma, \tau \in \Sigma_\lambda$, and $w \in \Theta$.\n
6
We next convert a 1ppda into a specific form, called an “ideal shape,” where the 1ppda takes a “push-pop-we construct another 1ppda which satisfies (2)–(3) of the ideal-shape conditions together with the extra condition that all 3–4 are related to the condition (5). For no-endmarker 1ppda’s, nonetheless, we can define the same notion

Proof of Lemma 4.2. Let $M = (Q, \Sigma, \{\$, \$\}, \Gamma, \Theta\Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ be any endmarker 1ppda with $|Q| = n$, $|\Gamma| = m$, and push size $e$. We first apply Lemma 4.1 to obtain another 1ppda that halts only on or
after reading $. We recall the notation used in the proof of Lemma 4.1. Notice that the obtained machine in the proof uses only a distinguished set \( Q^{(8)} \) of inner states after reading $ and that, when it halts, its stack becomes empty. Moreover, we recall the two halting states \( q'_{acc} \) and \( q'_{rej} \) from the proof. Here, we set \( Q_{acc} = \{ q'_{acc} \} \) and \( Q_{rej} = \{ q'_{rej} \} \). For readability, we also call the obtained 1ppda by \( M \) and assume that \( |Q| = 2n + 2 \) and \( |\Gamma| = m \). Let \( \Gamma_\lambda = \Gamma \cup \{ \lambda \} \). Starting with this new machine \( M \), we perform a series of conversions to satisfy the desired ideal-shape conditions. In the process of such conversions, we denote by \( M_i = (Q_i, \Sigma, \{ \{ \}, \Sigma \}, \Gamma_i, \delta_i, q_{i, 0}, Z_{i, 0}, Q_{i, acc}, Q_{i, rej}) \) the 1ppda with push size \( e_i \) obtained at stage \( i \in \{ 1 \} \).

To our description simpler, we further introduce the succinct notation \( \delta^*[q, \lambda, a \mid p, \lambda, w] \) where \( q \in Q - Q_{halt} \), \( p \in Q \), \( a \in \Gamma \), and \( w \in \Gamma^* \) for the total probability of the event that, starting in state \( q \) with stack content \( az \) (for an “arbitrary” string \( z \in \Gamma^+ \)), \( M \) makes a (possible) series of consecutive \( \lambda \)-moves without accessing any symbol in \( z \), and eventually reaches inner state \( p \) with stack content \( wz \). This notation is formally defined as \( \delta^*[q, \lambda, a \mid p, \lambda, w] = \sum_{r, b, u, w} \delta^{(t)}[q, \lambda, a \mid r, \lambda, bu] \delta(r, \lambda, b \mid p, u') \) if \( w \neq \lambda \), where \( b \in \Gamma \), \( r \in Q \), and \( w = u'u \in \Gamma^+ \).

Notice that, in particular, \( \delta^*[q, \lambda, a \mid p, \lambda, \lambda] = 0 \) follows.

By our definition of 1ppda’s, all computation paths must terminate eventually on every input \( x \). Thus, any series of consecutive \( \lambda \)-moves makes the stack height increase by no more than \( e|Q||\Gamma| \) because, otherwise, the series produces an infinite computation path. It thus follows that (*) \( \delta^*[q, \lambda, a \mid p, \lambda, w] \neq 0 \) implies \( |w| \leq e|Q||\Gamma| = e(n + 2)m \leq 2enm \).

(1) We first convert the original 1ppda \( M \) to another error-equivalent 1ppda, say, \( M_1 \) whose \( \lambda \)-moves are restricted only to pop operations; namely, \( \delta_1(q, \lambda, a \mid p, w) = 0 \) for all elements \( p, q \in Q \), \( a \in \Gamma \), and \( w \in \Gamma^+ \). For this purpose, we need to remove any \( \lambda \)-move by which \( M \) changes topos stack symbol \( \lambda \) to a certain nonempty string \( w \). We also need to remove all transitions of the form \( \delta(q, \lambda, Z_0 \mid p, Z_0) \), which violates the requirement of \( M_1 \) concerning pop operations. Notice that, once \( M \) reads $, it makes only \( \lambda \)-moves with inner states in \( Q^{(8)} \) and eventually empties the stack.

We define \( Q_1 = Q \cup Q^{(8)} \), \( Q_{1, acc} = Q_{acc} \cup \{ q^{(8)} \mid q \in Q_{acc} \} \), \( Q_{1, rej} = Q_{rej} \cup \{ q^{(8)} \mid q \in Q_{rej} \} \), and \( \Gamma_1 = \Gamma \cup \{ \lambda \} \), where \( Z_{1, 0} \) is a new bottom marker not in \( \Gamma \). Moreover, we set \( q_{1, 0} \) to be \( q_0 \). It then follows from the above definition of \( M_1 \) that \( |Q_1| = 2|Q| = 4(n + 1) \) and \( |\Gamma_1| = m + 1 \). By Statement (*), the push size \( e_1 \) of \( M_1 \) is at most \( 2enm \). The probabilistic transition function \( \delta_1 \) is constructed formally in the following subcases (i)–(iii). For any value of \( \delta_1 \) not listed below is assumed to be 0.

(i) Recall that the first step of any 1ppda must be a non-$\lambda$-move in general. At the first step, \( M_1 \) changes \( Z_{1, 0} \) to \( uZ_0Z_{1,0} \) (for an appropriate string \( u \) in \( (\Gamma'^-)*) \) so that, after this step, \( M_1 \) simulates \( M \) using \( Z_0 \) as a standard stack symbol but with no access to \( Z_{1,0} \) (except for the final step of \( M_1 \)). This process is expressed as follows.

1. \( \delta_1(q_{0, 0}, Z_{1,0} \mid p, uZ_0Z_{1,0}) = \delta(q_{0, 0}, Z_0 \mid p, uZ_0) \) for \( u \) satisfying \( uZ_0 \in \Theta_\Gamma \).

(ii) Assume that \( az \) is \( M \)'s stack content and \( M \) makes a (possible) series of consecutive \( \lambda \)-moves by which \( M \) never accesses any stack symbol in \( z \). Consider the case where \( M \) changes \( a \) to \( bu \) by the end of the series and, at reading symbol \( \sigma \in \Sigma \cup \{ \}$ \), \( M \) replaces \( b \) with \( u' \) at the next step in order to perform \( w (= u'u) \). In this case, we merge this entire process into one singular non-$\lambda$-move.

2. \( \delta_1(q, \sigma, a \mid p, w) = \sum_{r, b, u, w} \delta^*[q, \lambda, a \mid r, \lambda, bu] \delta(r, \sigma, b \mid p, u') \) if \( \sigma \in \Sigma \), \( q \notin Q^{(8)} \), and \( a \in \Gamma \), where \( w = u'u \in \Gamma^+ \) and \( b \in \Gamma \).

3. \( \delta_1(q, \$, a \mid p, w) = \sum_{r, b, u, w} \delta^*[q, \lambda, a \mid r, \lambda, bu] \delta(r, \$, b \mid p, u') \) if \( b \in \Gamma \), where \( w = u'u \in \Gamma^+ \) and \( b \in \Gamma \).

Since we obtain \( \delta_1(q, \sigma, a \mid p, \lambda) = \sum_{r, b} \delta^*[q, \lambda, a \mid r, \lambda, b] \delta(r, \sigma, b \mid p, \lambda) \), non-$\lambda$-moves of \( M_1 \) may contain pop operations. Lines 1–3 indicate that the new push size \( e_1 \) is exactly \( e + 1 \), and thus \( \Theta_\Gamma \subseteq \Gamma^{e1} \) follows.

Let us consider the case where, with a certain probability, \( M \) produces a stack content \( bz \) by the end of the series of \( \lambda \)-moves and then pops \( b \) at the next step without reading any input symbol. In this case, we also merge this entire process into a single $\lambda$-move of pop operation as described below.

4. \( \delta_1(q, \lambda, a \mid p, \lambda) = \sum_{r, b} \delta^*[q, \lambda, a \mid r, \lambda, b] \delta(r, \lambda, b \mid p, \lambda) \) for \( q \in Q - Q_{halt} \) and \( a \in \Gamma(e^(-)) \), where \( r \in Q \) and \( b \in \Gamma \).

(iii) Assume that \( M \) has already read $ but \( M \) is still in a non-halting state, say, \( q^{(8)} \) in \( Q^{(8)} \) making a series of consecutive \( \lambda \)-moves. Unless \( M \) reaches \( Z_0 \), similarly to Line 4., we merge this series of \( \lambda \)-moves and one pop operation into a single $\lambda$-move.

8
5. $\delta_1(q^{(8)}, \lambda, a \mid p^{(8)}, \lambda) = \sum_{r, b} \delta^*\gamma(q^{(8)}, \lambda, a \mid r^{(8)}, \lambda, b)\delta(r^{(8)}, \lambda, b \mid p^{(8)}, \lambda)$ for $a \in \Gamma(\neg), \gamma^{(8)} \notin Q_{halt}$, and $b \in \Gamma$, where $r \in Q$ and $b \in \Gamma$.

Once $M$ enters an inner state $p^{(8)}$ in $Q_{halt}$ at scanning $Z_0$, we remove $Z_0$ and enter the same halting state.

6. $\delta_1(q^{(8)}, \lambda, Z_0 \mid p^{(8)}, \lambda) = \delta^*\gamma(q^{(8)}, \lambda, Z_0 \mid p^{(8)}, \lambda, Z_0)$ for $q^{(8)} \notin Q_{halt}$ and $p^{(8)} \in Q_{halt}$.

(2) We next convert $M_1$ to another error-equivalent 1pda $M_2$ that conducts only the following types of moves: (a) $M$ pushes one symbol without changing the exiting stack content, (b) it replaces the topmost stack symbol by a (possibly different) single symbol, and (c) it pops the topmost stack symbol. We also demand that all $\lambda$-moves of $M_2$ are limited only to either (b) or (c). From these conditions, we obtain $w_2 = 2$.

To describe the intended conversion, we introduce two notations. The first notation $[w]$ represents a new stack symbol that encodes an element $w \in \Gamma_1 \leq \Sigma^c_1$. To make $M_2$ remember the topmost stack symbol of $M_1$, we introduce another notation $[q, a]$ indicating that $M_1$ is in inner state $q$ reading $a$ in the stack.

We set $Q_2 = [Q_1, \Gamma_1]$, $\Gamma_2 = \{[w] \mid w \in \Gamma_1 \leq \Sigma^c_1, \exists b \in \Gamma_1 \exists a \in \Gamma_1^* [bww \in \Theta_1] \} \cup \{[a] \mid a \in \Gamma_1\}$. Let $Z_{2,0} = [Z_{1,0}, \Gamma_0]$ denote a new bottom marker and let $q_{2,0} = [q_{1,0}, Z_{1,0}]$ denote a new initial state. It then follows that $|Q_2| = |Q_1| \leq 6nnm$ and $|\Gamma_2| \leq 4n|\Theta_1| + |\Gamma_1| \leq |\Gamma_1| \leq \epsilon(n + 2)m(2m)^{c(n + 2)m + 1} + 2m \leq 2emnm(2m)^{2emnm}$ since $|\Theta_1| \leq |\Gamma_1| \leq (n + 1)$.

(i) Consider the case where $M_1$ reads $(\sigma, a) \in \tilde{\Sigma} \times \Gamma$, substitutes $bu$ (with $b \in \Gamma$ and $u \in \Gamma^*$) for $a$, and enters inner state $p$. In this case, $M_2$ reads $(\sigma, [w])$ with a current topmost stack symbol of the form $[w] \in \Gamma_2$, changes it to $[u][w]$ (or $[u]$ if $u = \lambda$) in state $[q, a]$ and changes $[bu]$ to $[w]$ by entering inner state $[p, b]$. The function $\delta_2$ is formally defined as follows.

For $l \leq 2$, holds.

(3) We further convert $M_2$ to $M_3$ so that $M_3$ satisfies (2) and, moreover, there is no stationary operation that replaces any topmost symbol with a “different” single symbol. This is done by remembering the topmost stack symbol without writing it down into the stack. For this purpose, we use a new symbol of the form $[q, a]$ (where $q \in Q_2$ and $a \in \Gamma_2$) to indicate that $M_2$ is in state $q$, reading $a$ in the stack. Let $Q_3 = [Q_2, \Gamma_2]$, $q_{3,0} = [q_{2,0}, Z_{2,0}]$, and $\Gamma_3 = \Gamma_2 \cup \{[\lambda]\}$, where $Z_{2,0}$ is the new bottom marker of $M_3$. We then obtain $|Q_3| = |Q_2| + 2\leq 6nnm(2m)^{2emnm}$ and $|\Gamma_3| \leq 2emnm(2m)^{2emnm}$.

Similarely, to $\delta^*\gamma(q, \lambda, a \mid p, \lambda, w)$, we define $\delta^*_2(q, \lambda, a \mid p, \lambda, w)$ from $\delta_2$ to merge a series of consecutive $\lambda$-moves. If $M_2$ makes a series of $\lambda$-moves of changing $(q, a)$ to $(r, d)$ followed by a non-$\lambda$-move of changing $(r, d)$ to $(p, b)$ (resp., $(p, bd)$) while reading symbol $\sigma$, then $M_3$ reads $(\sigma, c)$ for a certain topmost symbol $c$, and changes its inner state $[q, a]$ to $[p, b]$ without changing $c$ (resp., with changing $c$ to $ac'$). In contrast, if $M_2$ changes to $(p, b)$, after a series of $\lambda$-moves, from $(q, a)$, then $\delta_3$ satisfies Condition 4. Note that a pop operation of $M_3$ is performed only at a certain $\lambda$-move.

Formally, $\delta_3$ is constructed as follows. Let $p, q \in Q_2$ and $a, b \in \Gamma_2, c \in \Gamma_3$, and $\sigma \in \Sigma$.

1. $\delta_3(q, \lambda, \sigma, c \mid p, [b, c]) = \sum_{r, d} \delta^*_2(q, \lambda, c \mid r, d)\delta(r, \sigma, d \mid p, b).
2. $\delta_3(q, \lambda, \sigma, c \mid [p, b], ac) = \sum_{r, d} \delta^*_2(q, \lambda, c \mid r, d)\delta(r, \sigma, d \mid p, bd).
3. $\delta_3(q, \lambda, \sigma, c \mid [p, b, c]) = \lambda = \sum_{r, d} \delta^*_2(q, \lambda, c \mid r, d)\delta(r, \lambda, d \mid p, \lambda).
4. $\delta_3(q, \lambda, \sigma, c \mid [p, b, c]) = \lambda = \sum_{r, d} \delta^*_2(q, \lambda, c \mid r, d)\delta(r, \lambda, d \mid p, b).

(4) We convert $M_3$ to $M_4$ that satisfies (3) and also makes only $\lambda$-moves of pop operations that follow only a (possible) series of pop operations. Let $Q_4 = Q_3 \cup \{p' \mid p \in Q_{halt}\}$ and $\Gamma_4 = \Gamma_3$. Let $q_{4,0} = q_{3,0}$ and $Z_{4,0} = Z_{3,0}$. It follows that $|Q_4| \leq 2|Q_3| \leq 4emnm(2m)^{2emnm}$ and $|\Gamma_4| \leq 4emnm(2m)^{2emnm}$. The probabilistic transition function $\delta_4$ is constructed as follows. A basic idea of our construction is that, when $M_3$ makes a pop operation after a certain non-pop operation, we combine them as a single move. As a new notation, we define $\hat{p} = p$ if $p \notin Q_{halt}$ and $\hat{p} = p'$ otherwise. Let $\sigma \in \Sigma_\lambda$. 

9
1. \( \delta_4(q, \sigma, a | \rho, \lambda) = \sum_r \delta_3(q, \sigma, a | r, a) \delta_3(r, \lambda, a | p, \lambda) \).
2. \( \delta_4(q, \sigma, a | \rho, p, \lambda) = \sum_r \delta_3(q, \sigma, a | r, ba) \delta_3(r, \lambda, b | p, \lambda) \) if \( \sigma \neq \lambda \).
3. \( \delta_4(q, \sigma, a | p, ba) = \delta_3(q, \sigma, a | p, ba) \) if \( \sum_{a \in Q_0} \delta_3(p, \lambda, b | s, \lambda) = 0 \) and \( \sigma \neq \lambda \).

We then obtain \( e_4 = 2 \).

(5) Finally, we set \( M_4 \) to be the desired 1ppda. It is clear that \( N \) satisfies all the requirements of the ideal-shape form. Thus, the proof of the lemma is completed.

It is important to note that the transformation of a general 1ppda to an error-equivalent 1ppda in an ideal shape is quite costly in terms of stack-state complexity. This high cost affects the overall cost of transforming one endmarker 1ppda to another error-equivalent non-endmarker 1ppda.

Before closing this section, we give a short remark. Steps (3) and (4) in the proof of Lemma 4.2 can be combined so that we can further remove any \( \lambda \)-move by introducing a new set of instructions. More precisely, we set a new transition function \( \delta \) as a map from \((Q - Q_{halt}) \times \Sigma \times (\Gamma \cup P(\Gamma - \{Z_0\})) \times Q \times \Gamma^*\) to \([0,1]\), where the notation \( \delta(q, \sigma, \{a_1, a_2, \ldots, a_k\} | p, \lambda) \) means that we repeat the pop operation as long as the topmost stack symbols are in \( \{a_1, a_2, \ldots, a_k\} \) and by verifying the validity of the guessing at the next step. If the guessing is incorrect, then \( N \) immediately enters both an accepting state and a rejecting state with the equal probability.

4.3 Usefulness of the Ideal-Shape Form

We have introduced to pushdown automata the notion of “ideal shape” form, which controls the behaviors of push and pop operations of the pushdown automata. To demonstrate the usefulness of this specific form in handling the machines, we intend to present a simple and quick example of how to apply the ideal-shape lemma (Lemma 12) for endmarker 1ppda’s.

**Proposition 4.3** The family of all languages recognized by endmarker 1ppda’s with bounded-error probability is closed under reversal.

**Proof.** Let \( L \) denote any language and let \( M = (Q, \Sigma, \{\varepsilon, \$\}, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej}) \) be any 1ppda that recognizes \( L \) with bounded-error probability. By the ideal shape lemma (Lemma 12), we can further assume that \( M \) is in an ideal shape. Hereafter, we construct another 1ppda \( N = (Q', \Sigma, \{\varepsilon, \$\}, \Gamma', \delta', q_0', Z_0, Q'_{acc}, Q'_{rej}) \) that takes \( x^R \) as an input and enters the same type of halting states as \( M \) does on the input \( x \) with the same error probability. A basic idea of the following construction is to reverse a computation of \( M \) step by step by probabilistically guessing the previous step of \( M \) and by verifying the validity of the guessing at the next step. If the guessing is incorrect, then \( N \) immediately enters both an accepting state and a rejecting state with the equal probability to cancel out this incorrect guessing.

Formally, we set \( Q' = (Q \times (\Sigma \cup \{\varepsilon, \$\})) \cup \{q_{acc}, q_{rej}\} \), \( Q_{acc} = \{q_{acc}\}, Q_{rej} = \{q_{rej}\} \), and \( q_0' = (q_0, \varepsilon) \). A new probabilistic transition function \( \delta \) is defined as follows. Let \( q, \sigma, \tau \in \Sigma \cup \{\varepsilon, \$\} \). Let \( \delta'((p, \tau), \tau, b | (q, \sigma, \lambda)) = \delta(q, \sigma, a | p, b), \delta'((p, \tau), r, c | (q, \sigma), a) = \delta(q, \sigma, a | p, \lambda), \delta'((p, \sigma), \lambda, b | (q, \sigma), \lambda) = \delta(q, \lambda, a | p, b), \) and \( \delta'((p, \sigma), \lambda, c | (q, \sigma), a) = \delta(q, \lambda, a | p, \lambda) \). In addition, for any halting state \( p \in \{q_{acc}, q_{rej}\} \), we define \( \delta'((q, \sigma), r, \varepsilon | (q, \sigma), c) = 1/2 \) if \( \sigma \neq r \).

It is not difficult to show that, for each \( \tau \in \{acc, rej\} \), \( p_{N,\tau}(x^R) = p_{N,\tau}(x) \) holds for any input \( x \in \Sigma^* \). Therefore, \( N \) correctly recognizes \( L^R \) with bounded-error probability.

5 From Endmarker 1ppda’s to No-Endmarker 1ppda’s

Following the preparatory steps of Section 4 let us prove Proposition 1.4. The proof of this proposition is composed of several key ingredients, each of which uses a different idea to place more restrictions on 1ppda’s. In particular, the proof will proceed in three stages by firstly removing \( \varepsilon \) (Section 5.1), secondly removing the last series of \( \lambda \)-moves after reading \( \$ \) (Section 5.2), and finally removing \( \$ \) (Section 5.3). In Section 5.4 we will combine these transformations and prove a technical lemma (Lemma 5.5), from which Proposition 1.4 follows.

5.1 Removing the Left Endmarker

We begin with removing the left endmarker \( \varepsilon \) from the 1ppda’s input tape. What we need here is to construct a 1ppda without \( \varepsilon \), which simulates a given endmarker 1ppda. If we had made the first set of moves “\( \lambda- \)
moves”, then we could have easily simulated the given machine. By our definition, however, any 1ppda cannot start with a λ-move in order to simulate the machine with ‡.

For clarity, we call by a no-left-endmarker 1ppda any 1ppda whose input tape uses only the right endmarker and a tape head starts at the leftmost symbol, instead of ‡, written on the input tape. In particular, when the input is λ, the left-most symbol becomes §.

**Lemma 5.1** Let M be any n-state 1ppda with stack alphabet size m. There is a 1ppda N that uses no left endmarker, has at most 2n states and at most (n + 1)m stack alphabet size, and is error-equivalent to M.

To obtain a precise stack-state complexity of transformation in Lemma 5.1, we provide an explicit construction of the desired machine. A basic idea of the construction of N is to remember the leftmost input symbol while the target machine M reads ‡ and makes a (possible) series of λ-moves.

**Proof of Lemma 5.1** We begin with a 1ppda M = (Q, Σ, {‡, §}, Γ, δ, q0, Z0, Q_{acc}, Q_{rej}) equipped with the two endmarkers with |Q| = n, |Γ| = m, and push size c. We want to construct its error-equivalent 1ppda N that uses no left endmarker ‡. We express N as (Q’, Σ, {§}, Γ’, δ’, q0, Z0, Q’_{acc}, Q’_{rej}) and hereafter describe each component of N in detail. Let x = x_1x_2⋯x_n denote any input string of length n over Σ.

Let Γ(−) = Γ − {Z_0} and Σ_λ = Σ ∪ {λ} and recall the notation [Σ, Γ] from Section 2.1. Define Q’ = Q ∪ {q | q ∈ Q}, Γ’ = Γ ∪ [Σ, Γ], and Θ_F’ = Θ_F ∪ [Σ, Γ]^2 ∪ [Σ, Γ]Z_0 ∪ [Σ, Γ]^2Z_0. We use the convention that, if a = λ, we set [σ, a] to be λ for any σ. Clearly, |Q’| = 2n and |Γ’| = (n + 1)m hold.

At the first step, N must read the leftmost symbol x_1 of the string x§ written on the input tape. During this first step as well as its (possible) subsequent λ-moves, N takes the same steps as M does on x‡x_1. For this purpose, N needs to remember x_1 as a part of its inner state until M actually reads x_1 after making a series of transitions (including λ-moves) related to ‡. In particular, while M makes a λ-move, N simulates this move without moving the tape head. Once M reads x_1 completely, N mimics M’s moves on x_1 without reading any new input symbol.

The probabilistic transition function δ’ is formally given below. Let p, q ∈ Q, σ ∈ Σ, ‡ ∈ Σ_λ, a, b ∈ Γ, w ∈ Γ^*, k ∈ N, and a_i ∈ Γ(−) ∪ {λ} for all i ∈ [k].

1. δ’(q_0, Z_0 | ‡, [σ, a_k]⋯[σ, a_1][σ, Z_0]Z_0) = δ(q_0, ‡, Z_0 | p, a_k ⋯ a_1 Z_0).
2. δ’(q_0, §, Z_0 | ‡, [§, a_k]⋯[§, a_1][§, Z_0]Z_0) = δ(q_0, ‡, Z_0 | p, a_k ⋯ a_1 Z_0).
3. δ’(q, ‡, λ, [σ, b] | ‡, [σ, a_k]⋯[σ, a_1][σ, b] Z_0) = δ(q, λ, b | p, a_k ⋯ a_1 b) if δ[q, λ, b] > 0.
4. δ’(q, ‡, λ, [σ, b] | ‡, λ) = δ(q, λ, b | p, λ) if δ[q, λ, b] > 0.
5. δ’(q, ‡, λ, [σ, b] | ‡, w) = δ(q, σ, b | p, w) if δ[q, λ, b] = 0.
6. δ’(q, ‡, σ, a | p, w) = δ(q, ‡, σ, a | p, w).
7. δ’(q, ‡, σ, a | p, w) = δ(q, ‡, σ, a | p, w).

Since N begins with reading the leftmost symbol x_1 instead of ‡, we need to remember x_1 until M actually reads x_1 after making a series of transitions on ‡ (in Line 1–2). While M makes a series of λ-moves, N simulates them without moving the tape head (in Line 3–4). After M makes the last λ-move in this series, N simulates M’s move of reading x_1 (in Line 5). Once M reads x_1, N mimics M’s behavior step by step (in Line 6–7).

Note that Line 2 is needed to deal with the case of the empty input. Moreover, Line 6 is needed because there are possibly bracketed symbols in the stack.

5.2 Removing the Final Series of λ-Moves

Next, let us consider a no-left-endmarker 1ppda M = (Q, Σ, {‡, §}, Γ, δ, q0, Z0, Q_{acc}, Q_{rej}). Clearly, if M enters halting states before reading the right endmarker § for all but finitely many inputs, then there is no trouble removing §. Hence, we hereafter assume that M reaches § for infinitely many inputs. Recall that, in our endmarker model, a 1ppda is permitted to make a series of λ-moves even after reading the right endmarker §. In the lemma stated below, we wish to remove this final series of λ-moves and force M to halt immediately after reading §. Furthermore, the lemma requires the condition that a given 1ppda is in an ideal shape. This condition guarantees that M never makes any pop operation at reading § (not including subsequent λ-moves).

**Lemma 5.2** Let M be any n-state 1ppda with stack alphabet size m using no left endmarker in an ideal shape. Assume that M enters a halting state along any computation path with a single accepting state (as well as a single rejecting state) only after emptying the stack. There is an error-equivalent no-left-endmarker
1ppda $P$ in an ideal shape that has at most $n(n+1)$ states and stack alphabet size $m2^n$ and also makes no $\lambda$-move after reading the right endmarker $\$. 

Lemma 5.3 is the most essential part of our proof of Proposition 4.4. Our proof of the lemma is inspired by [6] Lemma 2], in which Kaneps, Geidmanis, and Freivalds demonstrated how to convert each bounded-error unary 1ppda into an equivalent bounded-error “inputless” 1ppda, where a unary input alphabet. Since our setting (including our assumption) is different from [6] Lemma 2], we wish to provide the complete proof of the lemma. A key idea of Kaneps et. al. relies on a result of Nasu and Honda [3] Section 6] regarding a certain property of one-way probabilistic finite automata (or 1pfa’s) on reversed inputs. In essence, for any given 1pfa $M$ with no $\lambda$-move, Nasu and Honda constructed another 1pfa $N$ that accepts $x^R$ with the same probability as $M$ accepts $x$, assuming that we carefully choose which inner states of $N$ to start and terminate.

Since $M$ is in an ideal shape, any $\lambda$-move must pop a stack symbol. Thus, a series of pop operations (of the form $\delta_M(q, \lambda, \sigma | p, \lambda)$) after reading $\$ can be seen as a series of probabilistic transitions consuming each stack symbol from the topmost symbol to the bottom symbol. In other words, this process is completely described as an appropriate 1pfa taking a stack content as its input and making no $\lambda$-move. Such a 1pfa is described as $N = (Q, \Gamma^{(-)}, \{Z_0\}, \delta_N)$ with no initial state nor halting states, where $\Gamma^{(-)}$ is used as an input alphabet, $Z_0$ is treated as a new right endmarker, and $\delta_N(q, \sigma, p) = \delta_M(q, \lambda, \sigma | p, \lambda)$ for any tuple $(p, q, \sigma)$. We force $N$ to run until $Z_0$ is completely read. For our convenience, we call such a machine a free 1pfa $P$.

Given such a free 1pfa $N$, the notation $p_N(q, w, p)$ denotes the probability that $N$ starts in inner state $q$, reads $w$ completely, and then enters inner state $p$. Formally, for any $w = w_1w_2\ldots w_n \in (\Gamma^{(-)})^* \cup (\Gamma^{(-)})^*Z_0$, $p_N(q, w, p)$ is defined to be $\sum_{n \geq 1} \sum_{p_1, \ldots, p_n \in Q} (\prod_{i \in [n]} \delta_N(q_i, w_i, p_{i+1}))$ with $p_1 = q$ and $p_{n+1} = p$.

The acceptance/rejection probability of $M$ on input $x$ is given by the probability distribution produced by $M$ after completely reading $x\$ (without making $\lambda$-moves after reading $\$) multiplied by the probability of $N$’s processing (as its own input) the stack content generated by $M$ on $x$. However, to simulate $N$ properly, we need to read the stack content of $M$ from the bottom symbol to the top symbol: the “reverse” of the input given to $N$. What we need for the construction of the desired 1ppda $P$ in Lemma 5.3 is a free 1pfa $K$ that simulates $N$ on input $x$ but $K$ reads the reverse of $x$.

We thus introduce a supportive lemma, stated as Lemma 5.3. This lemma ensures that we can construct a free 1pfa $K$ that “mimics” the behavior of a given 1pfa $M$ by reading a “reversed” input if we choose appropriate inner states to start and terminate.

Lemma 5.3 Take $N$ and $p_N$ that are constructed from $M$ as above. There exists a free 1pfa $K = (Q_K, \Gamma^{(-)}, \{Z_0\}, \delta_K)$ that satisfies the following: for any pair $p, q \in Q$, there exist an inner state $s(p) \in Q_K$ and a set $T(q) \subseteq Q_K$ satisfying $p_N(q, w, p) = \sum_{r \in T(q)} p_K(s(p), w^R, r)$ for any $w \in (\Gamma^{(-)})^* \cup (\Gamma^{(-)})^*Z_0$, where $p_K$ is defined from $K$ similarly to $p_N$. Moreover, $|Q_K| = 2^{|Q|}$ holds.

Our proof of Lemma 5.3 closely follows the proof of Nasu and Honda [9] Theorem 6.1] to construct the desired $K$. A basic idea of Nasu and Honda is explained as follows. A free 1pfa’s transition can be described by a certain stochastic matrix, say, $U_N$ over inner states. To simulate this matrix on the reverse $x^R$ of input $x$, we need to construct a stochastic matrix $V$ that produces the same acceptance/rejection distribution as the transpose $U_N^T$ of $U_N$. Since the transpose $U_N^T$ may not be stochastic, we need to expand the original vector space to a larger vector space. Using a construction similar to Nasu and Honda’s, we can build the desired $K$ from $N$.

Proof of Lemma 5.3 Let $M = (Q, \Sigma, \{\$, \Gamma, \Theta, \delta, \theta, Z_0, Q_{acc}, Q_{rej})$ be a no-left-endmarker 1ppda in an ideal shape. Let the new notation $[x \in A]$ denote the truth value of the statement “$x \in A$”; namely, $[x \in A]$ equals 1 if $x \in A$, and 0 otherwise. Recall that $\Gamma^{(-)} \cup \{Z_0\}$ coincides with $\Gamma$. Letting $Q = \{q_1, q_2, \ldots, q_n\}$ with $q_1 = q_0$, we define $Q_K = P(Q)$ and express $Q_K$ as $Q_1, Q_2, \ldots, Q_{2^n}$ with $Q_1 = \emptyset$ and $Q_{2^n} = Q$. For each symbol $\sigma \in \Gamma$, we define $\alpha_{ij}^{(\sigma)} = \delta_N(q_j, \sigma | q_i)$ for any pair $i, j \in [n]$ and set $\alpha_i^{(\sigma)} = (\alpha_{i1}^{(\sigma)}, \ldots, \alpha_{in}^{(\sigma)})$ for any $i \in [n]$. Associated with $\alpha_{i}^{(\sigma)}$, we want to define a new vector $\gamma_i^{(\sigma)} = (\gamma_{k1}^{(\sigma)}, \ldots, \gamma_{k2^n}^{(\sigma)})$ for each index $k \in [2^n]$.

Hereafter, we explain how to define $\gamma_i^{(\sigma)}$ for any $k \in [2^n]$. As our preparation, we introduce three extra notations: $\min(\alpha_i^{(\sigma)})$ denotes the minimum positive entry in the vector $\alpha_i^{(\sigma)}$, namely, $\min(\alpha_{ij}^{(\sigma)}) | i \in [n], \alpha_{ij}^{(\sigma)} > 0$, $\text{unit}(\alpha_i^{(\sigma)})$ is the $\{0, 1\}$-vector expressing the truth values of the statements “$\alpha_{ij}^{(\sigma)} > 0$”, namely, $\langle \tilde{a}_1, \ldots, \tilde{a}_n \rangle$ for each $j \in [n]$, where $\tilde{a}_j = \alpha_{ij}^{(\sigma)} > 0$, and $e_j$ denotes the vector $(q_1 \notin Q_k, \ldots, q_n \notin Q_k)$ for each index $j \in [2^n]$. 

12
Given an arbitrary index $k \in [2^n]$, we define $\gamma_k(\sigma)$ from $\{\alpha_i(\sigma)\}_{i \in [n]}$ in the following recursive way. Recursively, we define $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(t)}$ as follows. Initially, we set $\alpha^{(1)} = \sum_{i \in Q_0} \alpha_i(\sigma)$ and choose a unique index $\ell_1 \in [2^n]$ satisfying $e_{\ell_1} = \text{unit}(\alpha^{(1)})$. Let us assume by induction hypothesis that $\alpha^{(i)}$ is already determined. Let $\alpha^{(i+1)} = \alpha^{(i)} - \min(\alpha^{(i)}\text{unit}(\alpha^{(i)}))$. If $\alpha^{(i+1)}$ is not the null vector, then we choose a unique index $\ell_{i+1} \in [2^n]$ satisfying $e_{\ell_{i+1}} = \text{unit}(\alpha^{(i+1)})$. When $\alpha^{(i+1)}$ becomes the null vector for a certain $t \in [n-1]$, we stop the recursive process and define $\gamma_k(\sigma) = 1 - \sum_{i \in [n]} \gamma_{k_i}(\sigma)$ and $\gamma_{kj} = 0$ for all $j$ not in $\{\ell_1, \ell_2, \ldots, \ell_t\}$. Finally, we set $\delta_K(Q_j, \sigma, q_j)$ to be $\gamma_k(\sigma)$. Moreover, we set $s(p) = \{p\} \in Q_K$ and $T(q) = \{A \in Q_K \mid q \in A\}$.

The desired equality $p_N(q, w, p) = \sum_{r \in T(q)} p_K(s(p), w^R, r)$ comes from the above definition of $K$. Note that $|Q_K| = |P(Q)| = 2^{|Q|}$.

Let us return to Lemma 5.2 and provide its proof with the help of Lemma 5.3.

**Proof of Lemma 5.2** Let $M = (Q, \Sigma, \{S\}, \Gamma, \Theta, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ be any no-left-endmarker 1pda in an ideal shape with $Q_{acc} = \{q_{acc}\}$ and $Q_{rej} = \{q_{rej}\}$. Our goal is to remove all $\lambda$-moves made by $M$ after reading $\$. Associated with the machine $M$, we take a free 1pfa $K = (Q_K, \Gamma^{(-)}, \{Z_0\}, \delta_K)$ and functions $s(\cdot), T(\cdot)$, and $p_K(\cdot)$ satisfying Lemma 5.3. Remember that $K$ can mimic a series of $\lambda$-moves of $M$ by an appropriate choice of inner states and that $K$ starts in initial state $q_0 = s(q_{acc})$.

Let $P = (Q', \Sigma, \{S\}, \Gamma', \Theta', \delta', q_0', Z_0, Q_{acc}, Q_{rej})$ denote the desired 1pda by which we want to simulate $M$ as well as $K$. The machine $P$ needs to remember both inner states of $M$ and $K$ in the form of $(q, q')$ during the intended simulation. When $M$ pushes a stack symbol, $P$ simulates its move and, simultaneously, it runs $K$ on this pushed symbol. Since $K$ uses a different set of inner states, we need to remember the changes of such states in its own stack. A stack symbol of the form $[a, r]$ indicates that $K$ is in inner state $r$ reading $a$.

Formally, we set $q_0' = s(q_{acc})$, $\Gamma' = [Q, Q_K]$, $Q' = Q \times K$, $Q_{acc}' = \{(p, p') \in Q' \mid p' \in T(p)\}$, and $Q_{rej}' = Q' - Q_{acc}'$. It then follows that $|Q'| \leq n + n^2 = n(n + 1)$ and $|\Gamma'| = |\Gamma||Q_K| = m2^n$. The probabilistic transition function $\delta'$ is constructed to satisfy the following conditions. We wish to define $\delta'$ in stages (1)--(3). Let $p, q, q', q'' 
 K, a, b \in \Gamma$, and $\sigma \in \Sigma \cup \{\lambda, \$\}.

1. The initial transition of $\delta'$ is defined as follows. Recall that the first step of $M$ cannot be a $\lambda$-move.

   1. $\delta'((q_0, q_0'), \sigma, Z_0 \mid (p, p'), Z_0, Z_0) = \delta(q_0, \sigma, Z_0 | p, Z_0)\delta_K(q_0', Z_0 | p')$ if $\sigma \neq \lambda$. 
   2. $\delta'((q_0, q_0'), \sigma, Z_0 \mid (b, q_0'), Z_0) = \delta(q_0, \sigma, Z_0 | p, bZ_0)\delta_K(q_0', Z_0 | b')$ if $\sigma \neq \lambda$. 

2. Consider the case where $M$ is in state $q$ reading input symbol $\sigma$ and stack symbol $a$ and $P$ is in state $(q, q')$ reading stack symbol $[a, r]$. If $M$ pushes $b$ into the stack and $K$ is in inner state $q'$ reading $[a, r]$ associated with $a$ (Line 3), then $P$ enters $(p, p')$ and pushes $[b, q']$ with the same probability multiplied by the extra probability that $K$ processes $b$ and enters $q''$ from $q'$. It is important to remember the inner state $q''$ of $K$ in the stack. If $M$ does not change the topmost stack symbol $a$ (Line 4), then $P$ only changes $(q, q')$ to $(p, q')$ since we do not need to simulate $K$'s step.

3. $\delta'((q, q'), \sigma, [a, r] \mid (p, p'), [b, q'] [a, r]) = \delta(q, \sigma, a | p, ba)\delta_K(q', b | p')$ for $\sigma \in \Sigma \cup \{\$\}$ if $\sigma \neq \lambda$.

4. $\delta'((q, q'), \sigma, [a, r] \mid (p, q') [a, r]) = \delta(q, \sigma, a | p, a)$ if $\sigma \neq \lambda$.

3. In contrast to (2), if $M$ pops a, then we need to “canceled out” the probability that $K$ processes a in state $r$. This is a crucial process in our simulation. If $K$ is a reversible machine, we can easily reverse the computation to nullify the effect of $\delta_K$ at the previous step. Since $K$ is generally not a reversible machine, nonetheless, we must remember the previous inner state $r$ and force $K$ to roll back to $r$. For this purpose, $P$ makes the following move.

5. $\delta'((q, q'), \sigma, [a, r] \mid (p, r), \lambda) = \delta(q, \sigma, a | p, \lambda)$.

When $M$ pops symbol $a$ (in Line 5), $P$ also pops $[a, r]$ with the same probability but rolls back the inner state of $K$ from $q'$ to its previous state $r$ to cancel out the probability that $K$ processes a in state $r$ and enters $q''$.

4. Assume that $M$ has already produced a stack content of the form $wZ_0$ and $M$ is now reading $. Notice that, by this moment, $P$ must have already simulated the process of $K$ on the input $(wZ_0)^R$. Recall that $K$ provides the acceptance/rejection probability that $M$ makes (a possible) series of consecutive $\lambda$-moves after reading $. When $M$ pops symbol $a$ with a certain probability at reading $\$, $P$ also pops $[a, r]$ with the same probability.

6. $\delta'((q, q'), $, $[a, r] \mid (p, p'), [b, q'] [a, r]) = \delta(q, $, $a | p, ba)\delta_K(q', b | p')$.

7. $\delta'((q, q'), $, $[a, r] \mid (p, q') [a, r]) = \delta(q, $, $a | p, a)$.
Lines 6–7 guarantee that, after reading $\$, $P$ does not make any $\lambda$-move.

By the above definition of $\delta'$, it is not difficult to show that $P$ correctly simulates $M$ with the same error probability. We remark that $P$ is also in an ideal shape.

5.3 Removing the Right Endmarker

Throughout Sections 5.1–5.2, we have constructed 1ppda’s that neither have the left endmarker nor makes any $\lambda$-move after reading the right endmarker $\$. Let $M$ be any of those 1ppda’s equipped with only the right endmarker $\$. In this final stage of our construction toward Proposition 5.2, we further remove $\$ from $M$. Notice that, without the endmarker $\$, the 1ppda’s cannot in general empty their stacks before halting.

Lemma 5.4 Let $M$ be any $n$-state 1ppda in an ideal shape with stack alphabet size $m$ but using no left endmarker; and assume that there is no $\lambda$-move after reading $\$. Moreover, we assume that $M$ does not alter the stack content at reading $. There exists an error-equivalent 1ppda $N$ with no endmarker using $2n+1$ states and stack alphabet size $(n+1)m$.

The proof of Lemma 5.4 is based on the following idea. Assume that we are reading an input symbol. We first simulate the corresponding move of $M$. We then predict an encounter of $\$ at the next move and, based on that prediction, we simulate the behavior of $M$ at reading $. In the next step, if the next input symbol is indeed $\$ as we have predicted, then we immediately decide either “accept” or “reject” according to the current inner state; otherwise, since our prediction is incorrect, we probabilistically “rewind” the last simulation and continue this whole process while reading the next input symbol. When $M$ is a reversible machine, this roll-back step can be easily constructed by reversing the last step of the machine. Since $M$ is in general not reversible, we must explore an idea used in the proof of Lemma 5.2 to remember the previous inner state taken right before the last step and then roll back to this state.

Proof of Lemma 5.4 Let $M = (Q, \Sigma, \{\}$, $\Gamma$, $\Theta$, $\delta$, $q_0$, $Z_0$, $Q_{acc}$, $Q_{rej}$) be any 1ppda in an ideal shape with no left endmarker. We assume that, on all inputs, $M$ makes no $\lambda$-move right after reading $\$ and, more importantly, $M$ does not alter any stack content at reading $. Note that, when reading $\$ at the final move, $M$ must make the final decision of whether it accepts or rejects an input without changing the stack content. To eliminate this final move, we must generate the transition probability of this particular move without reading $. This last transition must be of the form $\delta(p, \$, $a|\ r, a)$ for a certain triplet $(p, r, a)$ and we call this value $\delta(p, \$, $a|\ r, a)$ the decision probability at $(p, r, a)$. Our goal is to define a new no-endmarker machine $N = (Q', \Sigma, \Gamma', \Theta', \delta', q_0'$, $Z_0$, $Q'_{acc}$, $Q'_{rej}$) that simulates $M$ with the same error probability. Hereafter, we explain how to construct the desired machine $N$.

As for the key elements $Q'_{acc}$ and $Q'_{rej}$ of $N$, we set $Q'_{acc} = Q_{acc} \cup \{q_0\}$ and $Q'_{rej} = Q_{rej} \cup \{q_0\}$ otherwise. We further define $Q' = Q \cup \{\\} \\{ p \in Q \}$ and $\Gamma' = \Gamma \cup \{|, \}$ . Note that $|Q'| = 2n+1$ and $|\Gamma'| = m + nm = m(n+1)$. To simulate the first step of $M$, assuming that the next step is the final step, we produce the decision probability associated with this final step. Since this assumption may be false, we need to remember the assumed final move using the stack. At every step, we first cancel out the decision probability taken at the previous step. After rolling back to the previous step, we apply a new probabilistic transition and produce the next decision probability. To recover the previous decision probability, since the computation is not in general reversible, we need to remember a triplet $(p, r, a)$.

This process is formally expressed as follows. Let $\sigma \in \Sigma$ and $p \in Q$.

1. $\delta'(q_0, \sigma, Z_0 | p, [r, Z_0]Z_0) = \delta(q_0, \sigma, Z_0 | p, Z_0)\delta(p, \$, $Z_0 | r, Z_0)$.
2. $\delta'(q_0, \sigma, Z_0 | p, [r, a]Z_0) = \delta(q_0, \sigma, Z_0 | p, aZ_0)\delta(p, \$, $a | r, a)$ if $a \in \Gamma^{(-)}$.

In inner state $q$, $N$’s topmost stack symbol $[r, a]$ indicates that the last applied transition is of the form $\delta(p, \$, $a | r, a)$ for an inner state $p$ that entered from $q$ in a single step. We denote by $\mu_M(q, x, a | p, w)$ the probability that $M$ starts in state $q$ with a topmost stack symbol $a$, reads input $x$ completely, and reaches inner state $p$ with stack content $w$. Similarly, $\mu_N$ is defined using $\mu_M$. We want to enforce the following condition between $\mu_M$ and $\mu_N$.

$\text{Condition (*)}: \mu_N(q_0, x, Z_0 | p, [r, a]w) = \mu_M(q_0, x, Z_0 | p, aw)\delta(p, \$, $a | r, a)$ holds for an appropriately chosen quintuple $(p, x, r, a, w)$ with $x \neq \lambda$.

Assuming that Condition (*) is true, when $N$ reads off the entire input $x$, $N$ accepts and rejects $x$ with the same error probability as $M$ does. More precisely, if $M$ halts with probability $\gamma$ reaching state $r$ with
stack content $aw$, then $\gamma$ equals $\mu_M(q_0, x, Z_0 \parallel p, aw)\delta(p, $, $ a \mid r, a)$, and thus $N$ halts in inner state $p$ with probability $\gamma$.

In Lines 3–6 shown below, we follow the convention that, when either $\delta(q, $, $ a \mid r, a) = 0$ or $\delta(q, Z_0 \mid r, Z_0) = 0$, the left sides of the corresponding equations in Lines 3–6 must take value $0$. Let $\sigma \in \Sigma$, $a, b \in \Gamma$, $p, q \in Q$, and $w \in \Gamma^*$. If $M$ pushes $bw$ on the top of stack symbol $a$, we remember $w$ and place $[s, b]$ on the top of the stack (in Line 3).

3. $\delta'(q, \sigma, [r, a] \mid p, [s, b]w) = \delta(q, \sigma, a \mid p, bw)\delta(p, $, $ b \mid s, b)$ if $b \in \Gamma^{(−)}$ and $w \in (\Gamma^{(−)})^*$. 
4. $\delta'(q, \sigma, [r, a] \mid p, [s, b]) = \delta(q, \sigma, a \mid p, b)\delta(p, $, $ b \mid s, b)$ if $a \in \Gamma^{(−)} \cup \{\lambda\}$ and $\sigma \neq \lambda$.
5. $\delta'(q, \sigma, [r, Z_0] \mid p, [s, Z_0]) = \delta(q, \sigma, Z_0 \mid p, Z_0)\delta(p, $, $ Z_0 \mid s, Z_0)$ if $\sigma \neq \lambda$.

When $M$ pops $a$, $N$ cancels out the previous decision probability, pops $[r, a]$ instead, replaces any new topmost symbol, say, $b$ by $[s, b]$, and multiplies the corresponding decision probability (Lines 6–7).

6. $\delta'(q, \sigma, [r, Z_0] \mid p, \lambda) = 0$ holds for any $(q, \sigma, p)$. If Condition (*) is true, then $\delta'$ correctly simulates $\delta$ until $\$$ is read.

To complete the proof, we still need to verify Condition (*). We want to prove it by induction on the length of nonempty input $x$. If $|x| = 1$, then Lines 1–2 and 6–7 with $\sigma = \lambda$ imply Condition (*). Consider the case where our input is of the form $x \sigma$ for $\sigma \in \Sigma$. In this case, the value $\mu_N(q_0, x\sigma, Z_0 \parallel p, bw)$ with $\sigma \in \Gamma^{(−)}$ is the sum of the following terms, over all $q \in Q$, $(a) \mu_M(q_0, x, Z_0 \parallel q, aw)\delta(q, \sigma, a \mid p, b)$, $(b) \mu_M(q_0, x, Z_0 \parallel q, aw)\delta(q, \sigma, a \mid p, bw)$ with $w = bw$, and $(c) \mu_M(q_0, x, Z_0 \parallel q, abw)\delta(q, \sigma, a \mid p, \lambda)$. By induction hypothesis, we obtain $\mu_N(q_0, x, Z_0 \parallel p, [r, a]w) = \mu_M(q_0, x, Z_0 \parallel q, aw)\delta(p, $, $ a \mid r, a)$. Next, we calculate $\mu_N(q_0, \sigma, Z_0 \parallel q, [t, b]w)$. Here, we focus on term $(a)$, provided that the corresponding step of $\delta'$ is given by Line 4. It follows that

$$\sum_{r \in Q} \mu_N(q_0, x, Z_0 \parallel q, [r, a]w)\delta'(q, \sigma, [r, a] \mid p, [t, b])$$

$$= \sum_{r \in Q} \mu_M(q_0, x, Z_0 \parallel q, aw)\delta(p, $, $ a \mid r, a)\delta(q, \sigma, p, b)\delta(p, $, $ b \mid s, b)$$

$$= \mu_M(q_0, x, Z_0 \parallel q, aw)\delta(q, \sigma, p, b)\delta(p, $, $ b \mid s, b) \cdot \sum_{r \in Q} \delta(p, $, $ a \mid r, a)$$

Note that $\sum_{r \in Q} \delta(p, $, $ a \mid r, a) = 1$ holds because $M$ does not make any $\lambda$-move in state $p$ reading $. The other terms $(b)$–$(c)$ are in essence similarly handled. Therefore, Condition (*) is true, as requested. □

5.4 Proof of Proposition 1.4

Let us combine all transformations constructed in Sections 5.1–5.3 together with Section 4 to form the complete proof of Proposition 1.3. The proposition is actually an immediate consequence of a more technical lemma (Lemma 5.5). We thus intend to verify the lemma in this subsection.

Lemma 5.5 Given an $n$-state endmarker $1pda$ with stack alphabet size $m$ and push size $e$, there is an error-equivalent no-endmarker $1pda$ with $O(e^3n^6m^4(2m)^{5enm^2})$ states, stack alphabet size $O(e^3n^6m^6(2m)^{5enm^2}t^4)$, and push size 1, where $t = 2en^2m^2(2m)^{2enm}$. 

Proof. The proof of the lemma begins with taking an arbitrary $1pda$ $M = (Q, \Sigma, \{$, $\}, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$ with the two endmarkers. Let $|Q| = n$ and $|\Gamma| = m$. We apply Lemmas 1.1, 1.2 and then obtain a $1pda$ $M_1$ in an ideal shape with $Q_1$ and $\Gamma_1$ satisfying that $|Q_1| \leq 2en^2m^2(2m)^{2enm}$ and $|\Gamma_1| \leq 2enm(2m)^{2enm}$. We then apply Lemmas 5.1, 5.3 sequentially. Finally, we obtain a no-endmarker $1pda$ $M_2$ with $|Q_2| = O(e^3n^6m^4(2m)^{3enm})$ and $|\Gamma_2| = O(e^3n^6m^6(2m)^{5enm^2}t^4)$, where $t = 2en^2m^2(2m)^{2enm}$.

This completes the proof of the lemma. □
6 Quick Conclusion and Open Problems

Two different formulations have been used in textbooks and scientific papers to describe various models of pushdown automata in the past: pushdown automata with two endmarkers and those with no endmarkers. These two formulations have been well-known to be “equivalent” in recognition power for the deterministic and the nondeterministic models of one-way pushdown automata. Concerning the probabilistic model, the past literature [8, 4, 15, 16] used both formulations to obtain structural properties of pushdown automata but no proof of the “equivalence” between the two formulations (stated as the no endmarker theorem) has been published yet so far. This paper has given the formal proof of converting between the two formalisms and, more importantly, it has provided the first explicit upper bound on the stack-state complexity of transforming one formalism to another.

Among all questions that we have left unsettled in this paper, we wish to list three interesting and important questions concerning the no endmarker theorem.

1. It is desirable to determine the “exact” stack-state complexity of transforming endmarker 1ppda’s to error-equivalent no-endmarker 1ppda’s. If we weaken the requirement of “error-equivalence” by allowing small additional errors in the process of transformations, can we significantly reduce the stack-state complexity of transformation?

2. We may ask whether there exist much more concise 1ppda’s with no endmarker than what we have constructed in Section 5. More specifically, for example, is there any no-endmarker 1ppda that has only \((\epsilon \text{mm})^\Omega(1)\) states and stack alphabet size \(2^\epsilon \text{mm}^\Omega(1)\)?

3. As we have noted in Section 1.2, since the deterministic and the nondeterministic models of pushdown automata can be viewed as special cases of the probabilistic model, Lemma 5.5 provides an upper bound of the stack-state complexity of transformation for the deterministic and nondeterministic models. However, it seems likely that a much better upper bound can be achieved on the stack-state complexity of transformation.

References

[1] J. Eremondi, O. H. Ibarra, and I. McQuillan. Insertion operations on deterministic reversal-bounded counter machines. Journal of Computer and System Sciences 104 (2019) 244–257.
[2] S. Ginsburg and S. Greibach. Deterministic context free languages. Information and Control 9 (1966) 620–640.
[3] J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.
[4] J. Hromkovič and G. Schnitger. On probabilistic pushdown automata. Information and Computation 208 (2010) 982–995.
[5] O. H. Ibarra and I. McQuillan. The effect of end-markers on counter machines and commutativity. Theoretical Computer Science 627 (2016) 71–81.
[6] J. Kačeps, D. Geidmanis, and R. Freivalds. Tally languages accepted by Monte Carlo pushdown automata. In the Proceedings of the International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM’97), Lecture Notes in Computer Science, vol. 1269, pp. 187–195, 1997.
[7] H. R. Lewis and C. H. Papadimitriou. Elements of the Theory of Computation. Prentice-Hall, 1st edition (1981).
[8] I. I. Macarie and M. Ogiwara. Properties of probabilistic pushdown automata. Theoretical Computer Science 207 (1998) 117–130.
[9] M. Nasu and N. Honda. Fuzzy events realized by finite probabilistic automata. Information and Control 12 (1968) 284–303.
[10] K. Reinhardt. Counting and empty alternating pushdown automata. In J. Dassow, editor, Developments in Theoretical Computer Science: 7th IMYSC, number 6 in Topics in Computer Mathematics, Gordon and Breach Science Publishers S.A., pp. 123–132, 1992.
[11] G. Pighizzini and A. Pisoni. Limited automata and context-free languages. Fundamenta Informaticae 136 (2015) 157–176.
[12] T. Yamakami. Swapping lemmas for regular and context-free languages. Manuscript, arXiv:0808.4122, 2008.
[13] T. Yamakami. Oracle pushdown automata, nondeterministic reducibilities, and the hierarchy over the family of context-free languages. In the Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science, Lecture Notes in Computer Science, vol. 8327, pp. 514–525, 2014. A complete version at arXiv:1303.1717.
[14] T. Yamakami. Pseudorandom generators against advised context-free languages. Theoretical Computer Science 613 (2016) 1–27.
[15] T. Yamakami. One-way bounded-error probabilistic pushdown automata and Kolmogorov complexity (preliminary report). In the Proceedings of the 21st International Conference on Developments in Language Theory (DLT 2017), Lecture Notes in Computer Science, Springer, vol. 10396, pp. 353–364, 2017.

[16] T. Yamakami. Behavioral strengths and weaknesses of various models of limited automata. In the Proceedings of the 45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2019), Lecture Notes in Computer Science, Springer, vol. 11376, pp. 519–530, 2019.