How to Pop a Deep PDA Matters

Peter Leupold
Department of Mathematics, Faculty of Science
Kyoto Sangyo University
Kyoto 603-8555, Japan
eMail:leupold@cc.kyoto-su.ac.jp

Abstract

Deep PDA are push-down automata whose push-down alphabet is divided into terminal and non-terminal symbols. In the push-down, not only the top-most symbol can be read, but any non-terminal up to a given depth –counting only non-terminals– can be read and also replaced. This way the computational power is increased compared to conventional PDAs.

We study some variations of the original definition. Allowing pop operations to change the automaton's state increases the computational power. The same is true for admitting transitions that delete non-terminals. In the latter case we even present a computationally complete variant of deep push-down automata. As these investigations are only a first glance at the new variants, we conclude with a list of open problems.

1 Introduction

It is a classical result in Formal Language Theory, that the language class generated by context-free grammars is equal to the class of languages accepted by push-down automata (PDAs). This is usually shown by providing algorithms that convert one construct into the other. When looking at the push-down automaton that results from a context-free grammar in the common constructions, we can see that the push-down alphabet can be divided into two classes: symbols corresponding to the grammar's terminals, and symbols corresponding to the grammar's non-terminals. The terminals are simply matched against the input, i.e. they are popped from the top of the push-down, when they match the current symbol on the input tape. The non-terminals, in contrast, are expanded according to the productions of the grammar.

This observation inspired Meduna to introduce deep push-down automata (deep PDAs), where this distinction into two classes of symbols on the push-down is formalized. Terminals can only be popped from the push-down, non-terminals can be expanded in the way of a context-free rewriting rule. The essential feature of these automata is that the expansion of non-terminals can not only be done at the top of the push-down, but up to a certain depth, counting only non-terminals. If this depth is one, these devices are equivalent to conventional PDAs. With every increase in depth, however, we obtain a new class of languages, containing the ones of lesser depth. Thus these deep PDAs induce an infinite hierarchy of languages between

the context-free and context-sensitive languages. This hierarchy is equal to the one induced by state grammars [3].

In the definition of deep PDAs a few choices were made that are not obvious. For example, the operation of popping terminals cannot change the state of the automaton. Further, already Meduna states the question what the effect of admitting rules deleting non-terminals would be — in some definitions of context-free grammars these are admitted, although it is easy to prove that they are unnecessary and can be eliminated without changing the generative power. In some sense, deleting a non-terminal is the same as popping it. These are the two choices that we investigate here. Mainly, we explore what the effect of making the choice different from the one in the original definition would be.

2 Definitions

Terms and notation from general formal language theory, logic and set theory are assumed to be known to the reader and can be looked up in standard textbooks like the one by Harrison [1]. The length of a string w we denote by $|w|$, and $|w|_U$ is the number of occurrences of symbols from the set U in w.

With this, we already come to the central definition in our context, the one of deep push-down automata. In our notation we follow the article by Meduna [3].

Definition 1. A *deep pushdown automaton (deep PDA)* is defined to be a 7-tuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, where Q is a finite set of states, Σ is the input alphabet, and Γ is the pushdown alphabet; Σ is a subset of Γ, there is a bottom symbol $\#$ in $\Gamma \setminus \Sigma$. $s \in Q$ is the start state, $S \in \Gamma$ is the start pushdown symbol, $F \subset Q$ is the set of final states. The transition relation R is a finite set that contains elements of $(I \times Q \times (\Gamma \setminus (\Sigma \cup \{\#\}))) \times (Q \times (\Gamma \setminus \{\#\})^+)$ and of $(I \times Q \times \{\#\}) \times (Q \times (\Gamma \setminus \{\#\})^* \{\#\})$; here I is a a set of integers of the form $\{i : 1 \leq i \leq n\}$ for some n.

For the interpretation of this tuple as an automaton we need to describe its behaviour, which is done mainly by defining its transition relations on its set of configurations. A configuration of a deep PDA is a triple from $Q \times \Sigma^* \times (\Gamma \setminus \{\#\})^* \#$; this means that it is determined by the current state, the remaining contents of the input tape, and the contents of the push-down. For transitions $((m, p, A), (q, u))$ we will normally use the more suggestive notation $mpA \to qu$.

M pops its pushdown from x to y, symbolically written as $x \ {}_p\Rightarrow \ y$, if $x = (q, au, az), y = (q, u, z)$, where $a \in \Sigma, u \in \Sigma^*, z \in \Gamma^*$. *$M$ expands* its pushdown from x to y, symbolically written as $x \ {}_e\Rightarrow \ y$, if $x = (q, w, uAz), y = (p, w, uvz), mqA \to pv \in R$, where $q, p \in Q, w \in \Sigma^*, A \in \Gamma, u, v, z \in \Gamma^*$, and $|u|_{\Gamma \setminus \Sigma} = m - 1$. To express that M makes $x \ {}_e\Rightarrow \ y$ according to $mqA \to pv$, we write $x \ {}_e\Rightarrow \ y \ [mqA \to pv]$. We say that $mqA \to pv$ is a *rule of depth m*; accordingly, $x \ {}_e\Rightarrow \ y \ [mqA \to pv]$ is an *expansion of depth m*. M makes a *move* from x to y, symbolically written as $x \Rightarrow y$, if M makes either $x \ {}_e\Rightarrow \ y$ or $x \ {}_p\Rightarrow \ y$.

If $n \in I$ is the minimal positive integer such that each of M's rules is of depth n or less, we say that *M is of depth n*, symbolically written as ${}_nM$. In the standard manner, extend ${}_p\Rightarrow, {}_e\Rightarrow$, and $\Rightarrow$ to ${}_p\Rightarrow^m, {}_e\Rightarrow^m$, and $\Rightarrow^m$, respectively, for $m \geq 0$;

then, based on ${}_p\Rightarrow^m$, ${}_e\Rightarrow^m$, and $\Rightarrow^m$, define ${}_p\Rightarrow^+$, ${}_p\Rightarrow^*$, ${}_e\Rightarrow^+$, ${}_e\Rightarrow^*$, $\Rightarrow^+$, and $\Rightarrow^*$. Let M be of depth n, for some $n \in I$. We define the *language accepted by* ${}_nM$, $L({}_nM)$, as $L({}_nM) = \{w \in \Sigma^* | (s, w, S\#) \Rightarrow^* (f, \varepsilon, \#)$ in ${}_nM$ with $f \in F\}$. In addition, we define the *language that M accepts by empty pushdown,* $E({}_nM)$, as $E({}_nM) = \{w \in \Sigma^* | (s, w, S\#) \Rightarrow^* (q, \varepsilon, \#)$ in ${}_nM$ with $q \in Q\}$.

For every $k \geq 1$, ${}_{deep}\mathbf{PD}_k$ denotes the family of languages defined by the deep pushdown automata of depth i, where $1 \leq i \leq k$. Analogously, ${}^{empty}_{deep}\mathbf{PD}_k$ denotes the family of languages defined by the deep pushdown automata of depth i by empty pushdown, where $1 \leq i \leq k$. **CF**, **CS**, and **RE** denote the families of context-free, context-sensitive, and recursively enumerable languages, respectively.

3 Popping Without Change of the State

With the original definition of deep PDA as given in the preceding section, computations are in principle free to alternate between steps popping terminals and ones expanding non-terminals. However, if there is an accepting computation for a word, there is always a computation that first does all the expansions and only then starts to pop. Already the fact that expanding transitions do not read any input suggest that this is possible.

Lemma 2. *If a deep PDA M accepts a word w, then there is an accepting computation of M on w, which performs only pops after the first pop operation.*

Proof. Let γ be a computation of a deep PDA M. If γ already fulfills our lemma's conditions there is nothing to show. Otherwise there exists a sequence of two configurations in γ such that the first one is a pop operation, the following one an expansion of a non-terminal. Let the former pop the symbol a, and let the latter be the transition $nqA \to pU$. This means that the configuration just before these two transitions must have the symbol a on the top of the push-down. Since the pop does not change the state or introduce or change non-terminals, q is already the current state and A is the n-th non-terminal in the push-down. Therefore $nqA \to pU$ can also be applied before the pop. This, on the other hand, does not change the fact that a is the top-most symbol on the push-down, and thus it can be popped now, because the pop operation is defined for all combinations of terminals and states. The resulting configuration is the same as for popping first and then applying $nqA \to pU$. Summarizing, we can obtain another computation γ' where the pop and the transition are exchanged, but the result is the same as for γ.

Since this process can be iterated, all the pop operations can be moved to the end of the computation, and this can be done for any computation. □

This suggests a normal form for deep PDAs, where there is only one final state, only for this state the pop operations are defined, and no other transitions are defined for it, that is it cannot be left once it has been entered.

4 Popping With Change of the State

The definition of deep PDAs was inspired by the way the standard conversion of context-free grammars to PDAs works. In this, the grammar's terminals are only put on the push-down store and are then matched against the input. Although this is a normal transition for a PDA, in this construction it is usually not necessary to change the state in this type of transition. This is why this possibility was not included in the definition of deep PDAs. However, the central idea of deep PDAs seems to be the possibility to change non-terminals deep in the push-down; changing the state during popping is perfectly consistent with this. Therefore we now change the definition a little and then look whether this changes the computational power. The main differences to the original definition are that

- the pop operation is not defined automatically for all states, and
- when popping a terminal symbol, the state can be changed.

Definition 3. A *deep pushdown automaton with state-changing pop* is a deep pushdown automaton $(Q, \Sigma, \Gamma, R, s, S, F)$, which has the additional type of transition from the set $Q \times \Sigma \times Q$. Further, the pop operation is not defined implicitly.

The class of languages accepted by deep pushdown automata with state-changing pop of depth i is denoted by ${}_{deep}\mathbf{pPD}_i$

The interpretation of the new kind of transition is that for configurations $x = (q, au, az)$ and $y = (p, u, z)$, we have $x \ {}_p\Rightarrow\ y$ where $a \in \Sigma$, $u \in \Sigma^*$, $z \in \Gamma^*$, if (q, a, p) is a transition of the automaton.

Recalling that ${}_{deep}\mathbf{PD}_1 = \mathbf{CF}$, we now show that at depth one this new way of popping increases computational power compared to the original definition.

Proposition 4. *For all $i \geq 1$ the language $\{(wc)^i : w \in \{a, b\}^*\}$ can be accepted by a deep PDA with state change on pops of depth one.*

Proof. We construct an automaton for the language $\{(wc)^3 : w \in \{a, b\}^*\}$. It should then be clear how to extend the construction to languages consisting of w^i for $i > 3$. The state set is $\{s, p, q, p^2, p^3\} \cup \{p_x^2, p_x^3 : x \in \{a, b, c\}\}$, the non-terminals are S and A, p is the only final state. The rules are the following:

$1sS$	$\rightarrow$	qA	initiate
$1qA$	$\rightarrow$	qaA	write a random word w over $\{a, b\}$
$1qA$	$\rightarrow$	qbA	
$1qA$	$\rightarrow$	p^2cA	write c
$p^2(a)$	$\rightarrow$	p_a^2	pop a and remember it
$p^2(b)$	$\rightarrow$	p_b^2	
$1p_a^2A$	$\rightarrow$	p^2aA	copy a to the second copy of w
$1p_b^2A$	$\rightarrow$	p^2bA	
$p^2(c)$	$\rightarrow$	p_c^2	the end of the first copy of w is reached
$1p_c^2A$	$\rightarrow$	p^3cA	

$p^3(a)$	$\rightarrow$	p_a^3	copy a to the third copy of w
$p^3(b)$	$\rightarrow$	p_b^3	
$1p_a^3A$	$\rightarrow$	paA	
$1p_b^3A$	$\rightarrow$	p^3bA	
$p^3(c)$	$\rightarrow$	p_c^3	the end of the second copy of w is reached
$1p_c^3A$	$\rightarrow$	pc	no more non-terminals on the push-down
$p(a)$	$\rightarrow$	p	pop the third copy of w
$p(b)$	$\rightarrow$	p	
$p(c)$	$\rightarrow$	p	

The first four rules generate a random word from the language $\{a,b\}^*c$. After this, there is no more choice for the automaton. The states that pop can only pop the terminal on the top of the push-down. On the other hand there is always just one transition for the states that expand non-terminals. So the remainder of the computation is deterministic. The superscripts of states indicate the number of the copy of w that is currently written on the push-down, the subscripts remember, which terminal has been popped. Looking at an example computation should make it clear that (and how) exactly the words from $\{(wc)^3 : w \in \{a,b\}^*\}$ are accepted.

For constructing an automaton for languages $\{(wc)^i : w \in \{a,b\}^*\}$ with i greater than 3, simply more copying phases with states p_x^4, p_x^5, etc. need to be added, and the final state p needs to be accessed only after the last one of these. □

Standard pumping arguments show that the languages accepted are not context-free for $i \geq 2$ and therefore they are not accepted by deep push-down automata of depth one either.

The immediate question raised by the result of this section is whether similar languages without the letter c can be accepted by the same classes of device. c serves as the separator between different copies of the same word, thus it signals to the automaton that another copy has been completely read (and copied). Therefore it is essential to the way the automata work and it is not clear how it could be eliminated.

Note also that Proposition 4 suggests that something in the way of Lemma 2 will not hold for deep pushdown automata with state-changing pop.

5 Deleting Non-Terminals

Already in the seminal article on deep push-down automata, Meduna raised the question what the effect of admitting deleting rules would be. Without this possibility, every non-terminal on the push-down must eventually be rewritten to a terminal; the terminal must then be matched against the input. Thus in an accepting computation, the push-down can never contain more symbols than the word to be accepted. It is intuitively clear that this way only context-sensitive languages can be accepted. With deleting rules, the push-down can be expanded without limit and then be shrunk. The only question is, whether this really results in an increase in power. Before addressing this matter, we first define the deep push-down automata admitting deleting rules.

Definition 5. A *deleting deep pushdown automaton* is a deep pushdown automaton whose expanding rules can have empty right sides.

The class of languages accepted by deleting deep pushdown automata of depth i is denoted by ${}_{deep}\mathbf{dPD}_i$.

We now come back to the question raised at the end of the preceding section. In a straight-forward way we can delete the c from the language accepted there by replacing it with a non-terminal during the computation. This non-terminal is then simply deleted where the c was popped.

Proposition 6. *For all $i \geq 1$ the language $\{w^i : w \in \Sigma^+\}$ can be accepted by a deleting deep pushdown automaton of depth* 2.

Proof. We construct an automaton for the language $\{w^3 : w \in \Sigma^+\}$. It should then be clear how to extend the construction to languages consisting of w^i for $i > 3$. Further, we will restrict the alphabet to the size of two with $\Sigma = \{a, b\}$; also here the generalization to larger alphabets will be straight-forward.

$1sS$	$\rightarrow$	qA	initiate
$1qA$	$\rightarrow$	qaA	write a random word w over $\{a, b\}$
$1qA$	$\rightarrow$	qbA	
$1qA$	$\rightarrow$	p^2CA	write C
$p^2(a)$	$\rightarrow$	p_a^2	pop a and remember it
$p^2(b)$	$\rightarrow$	p_b^2	
$2p_a^2A$	$\rightarrow$	p^2aA	copy a to the second copy of w
$2p_b^2A$	$\rightarrow$	p^2bA	
$1p^2C$	$\rightarrow$	p_C^2	the end of the first copy of w is reached
$1p_C^2A$	$\rightarrow$	p^3CA	
$p^3(a)$	$\rightarrow$	p_a^3	copy a to the third copy of w
$p^3(b)$	$\rightarrow$	p_b^3	
$2p_a^3A$	$\rightarrow$	paA	
$2p_b^3A$	$\rightarrow$	p^3bA	
$1p^3C$	$\rightarrow$	p_C^3	the end of the second copy of w is reached
$1p_C^3A$	$\rightarrow$	p	no more non-terminals on the push-down
$p(a)$	$\rightarrow$	p	pop the third copy of w
$p(b)$	$\rightarrow$	p	

This set of transitions is very similar to the one for the automaton in the proof of Proposition 4. The only differences are that the terminal c from there is replaced by the non-terminal C here, and no C is written after the last copy of w. Further, the depth for some transitions has been changed to two, because now instead of c we have C above A on the push-down. But the major difference is that transitions $1p^2C \rightarrow p_C^2$ and $1p^3C \rightarrow p_C^3$ delete non-terminals instead of popping terminals. In this way, there is a border between the copies of w, which consists of a non-terminal rather than a terminal and therefore it is not visible in the accepted word. □

Now we turn our attention to the question what happens if we drop the bound on the depth. Meduna showed that without deleting rules this still results in a subset of

the context-sensitive languages. Of course, there are several ways in which to drop the bound on the depth of non-terminals that can be rewritten. We will drop the specification for the depth for some rules. This means there are rules of the form we already know, but also ones not specifying the depth. For the latter ones, one could apply rewriting steps on an arbitrary occurrence of the given non-terminal, or for example on the left-most one.

We do not fix this option, because either one will work for the proof of Proposition 7; all rules without specification of the depth will rewrite only non-terminals that occur just once in the push-down. Further possibilities, like rewriting all occurrences in parallel, are not considered. So the class of languages accepted by deleting deep push-down automata without bound on the depth is denoted by ${}_{deep}\mathbf{dPD}$, and we now sketch a proof for the fact that these devices are equivalent to Turing Machines.

Proposition 7. ${}_{deep}\mathbf{dPD} = \mathbf{RE}$.

Proof. [Sketch] We have seen in the proof of Proposition 4 how state-change in the pop operation can be used to copy a word on the push-down to a deeper location. If we have rules deleting non-terminals, we can copy words of non-terminals in the same way. With slight modifications, this copying can simulate one step of a Turing Machine working on the given word. Therefore, iteration of the process can simulate the work of a Turing Machine. So for any given Turing Machine M, we can run a simulation with a deep PDA and accept the same language as the original machine. The important steps are:

1. Generate a random word on the top of the push-down, followed by some symbol marking the end of the used part of the tape. The input is encoded such that for example a word *abaacb* is represented as $[z_0\frac{a}{a}]\frac{b}{b}\frac{a}{a}\frac{a}{a}\frac{c}{c}\frac{b}{b}$. Here z_0 is the starting state of M, the brackets show the head's position. The work of M will be simulated on the upper row of letters, the lower one will remain unchanged to remember the input word.

2. Simulate the work of M. For example, from the initial configuration given above, a step to the right writing c and changing the state to z_1 results in $\frac{c}{a}[z_1\frac{b}{b}]\frac{a}{a}\frac{a}{a}\frac{c}{c}\frac{b}{b}$.

   New space is created by introducing new symbols that have ϵ on the lower tape. For example, from the initial configuration given above, a step to the left writing c and changing the state to z_2 results in $[z_2\frac{c}{\epsilon}]\frac{a}{a}\frac{b}{b}\frac{a}{a}\frac{a}{a}\frac{c}{c}\frac{b}{b}$.

3. If the computation of M halts in an accepting state, copy the halting configuration to the terminals that are given on the second row and match these against the input by simple pop operations.

So if there exists an accepting computation of M on a word w, then there exists a computation of the deep PDA constructed which generates w on the push-down, simulates M's computation, and then accepts w on the input tape. On the other hand, if there exists no accepting computation of M for a given word w, then no computation of the deep PDA will ever write w on the push-down in terminals, and

thus this word will never be accepted. Thus the two devices are equivalent, since obviously a Turing Machine can simulate our deep PDA.

We will now get into more detail about how the simulation above is implemented on the deep PDA. Let the state set of the Turing Machine to be simulated be Z, its set of transitions R. The push-down alphabet is

$$\Gamma := \{[\frac{x}{y}], [z\frac{x}{y}] : z \in Z, x, y \in \Sigma\} \cup \{S, \#\} \cup \Sigma.$$

Notice that in the informal description above we have left away the brackets around the non-terminals $[\frac{x}{y}]$. The set of states will not be defined explicitly, it will simply contain exactly the states that are used in the definition of the transitions that follow.

$$\begin{array}{lcll} sS & \rightarrow & s[z_0\frac{x}{x}]A & \text{for all } x \in \Sigma \cup \{\epsilon\} \\ sA & \rightarrow & s[\frac{x}{x}]A & \text{for all } x \in \Sigma \\ sA & \rightarrow & qBA & \end{array}$$

The copying of the Turing Machine's tape must be done with a delay of one, because in the case of a move to the left the head must end up one symbol more to the left. Therefore one symbol must be remembered in the states. The symbol B only has the function of separating different copies of the Turing Machine tape.

In general, the copying is done by rules of the following type, where the read symbols are buffered in a first-in, first-out memory of size two in the states' subscripts.

$$\begin{array}{lcll} 1q_{[x/y]}[\frac{r}{s}] & \rightarrow & q_{[x/y][r/s]} & \text{for all } x, y, r, s \in \Sigma \cup \{\epsilon\} \\ q_{[x/y][r/s]}A & \rightarrow & q_{[r/s]}[\frac{x}{y}]A & \end{array}$$

The first type of transition pops the top-most push-down symbol and puts it in the buffer of the state. Then the second type of transition takes the older content of the buffer and puts it on the push-down next to A, which always marks the place in the push-down to where the copy process writes. So states with one tape cell in the buffer always pop, while those with two states in the buffer always have to write one symbol out of the buffer.

The only spot where this process has to be done slightly differently is the position of the Turing Machine's head. We only define as an example the deep PDA's transitions simulating a Turing machine step, which in the state z, reading the symbol a moves left, goes to state z' and writes the symbol b. From this it should be clear how to define these transitions in general.

$$\begin{array}{lcll} 1q_{[x/y]}[z\frac{a}{r}] & \rightarrow & q_{[z'/x/y][b/r]} & \text{for all } x, y, r \in \Sigma \cup \{\epsilon\} \\ q_{[x/y][b/r]}A & \rightarrow & q_{[b/r]}[z'\frac{x}{y}]A & \end{array}$$

Essentially the process is the same as simple copying, only the z is read, too, and z' is written to the tape cell already in the buffer.

Finally, the finite state control of our automaton can, whenever it writes a non-terminal of the form $[z\frac{x}{y}]$, know whether further moves of the Turing Machine will be possible, or whether it will stop. If it stops and is in a final state, then the input word has been accepted. In a last copying step, all non-terminals $[\frac{x}{y}]$ are rewritten to y, where this is not equal to ϵ. Since this second tape has not been altered during

the simulation of the Turing Machine, we then have the word w on the push-down and can match it against the input with pop operations.

For any given Turing Machine, if it accepts a word w, there exists a computation of the corresponding deep push-down automaton constructed here, which first generates randomly this word w, then simulates the Turing Machine's computation, and accepts. On the other hand, this deep push-down automaton cannot accept any word not accepted by the Turing Machine, because clearly any computation leading to non-terminals on the stack must pass through some accepting configuration of the Turing Machine. □

6 Open Problems

Our investigations have only shed a few spotlights on the wide range of questions that arise around the presented variations in the definition of deep PDAs.

In the article introducing Deep PDAs, Meduna showed that every class PD_i is properly included in the class PD_{i+1}, which can go one level deeper in the push-down. The method of proof was to show that each of these classes equals the class of languages generated by a corresponding class of state grammars. It is known that the state grammars induce an infinite hierarchy of classes of languages between the classes of context-free and context-sensitive languages. This was shown by Kasai, who employed rather complicated languages to separate these classes [2]. As described at the end of Section 4, looking at deep PDAs, languages $\{w^i : w \in \Sigma^+\}$ also seem to be good, simpler candidates for this. A close analysis of push-down behaviour might enable us to show this.

Open Problem 1. $\{w^i : w \in \Sigma^+\} \in PD_i$ *but* $\{w^i : w \in \Sigma^+\} \notin PD_{i-1}$*?*

Proposition 4 shows that with a state change in the popping operation languages can be accepted at depth one that cannot be accepted at depth one by conventional deep PDA. Thus it is not clear whether the hierarchy of language classes corresponding to the one induced by deep PDAs of different depth is also a true hierarchy, or whether it collapses.

Open Problem 2. *Is* ${}_{deep}\mathbf{pPD}_{i+1} \setminus {}_{deep}\mathbf{pPD}_i$ *non-empty for all* $i \geq 1$*?*

In Proposition 6 we need deleting deep push-down automata of depth two. We suspect that depth one does not increase computational power compared to conventional push-down automata.

Open Problem 3. *Is* ${}_{deep}\mathbf{dPD}_1 = \mathbf{CF}$*?*

Also for deleting deep push-down automata it is interesting whether they induce an infinite hierarchy.

Open Problem 4. *Is* ${}_{deep}\mathbf{dPD}_{i+1} \setminus {}_{deep}\mathbf{dPD}_i$ *non-empty for all* $i \geq 1$*?*

Proposition 7 shows that the possibility to delete non-terminals increases the computational power in the case of unbounded depth. However, the proof does not

give any indication as to whether the same is true with a bound on depth. It is clear that ${}_{deep}\mathbf{PD}_i$ is always a subset of ${}_{deep}\mathbf{dPD}_i$, but whether this inclusion is proper is not obvious, though we strongly suspect that it is.

Open Problem 5. *Is* ${}_{deep}\mathbf{PD}_i$ *a proper subset of* ${}_{deep}\mathbf{dPD}_i$ *for all* $i \geq 1$*?*

Over one-letter alphabet, it is known that the classes of regular and context-free languages coincide. Comparing finite automata with push-down automata, this means that the push-down store is of no use at all in the sense that its absence or presence have no influence on the computational power. This raises the question whether the different types of push-down stores introduced here make a difference over one-letter alphabet or not.

Open Problem 6. *Which deep push-down automata can accept non-regular languages over one-letter alphabet?*

If we allow popping non-terminals, we can easily obtain non-regular languages.

Example 8. Consider the deep PDA with the following transitions and the corresponding alphabets and state set.

$$\begin{array}{lcl} 1sS & \to & qaBA \\ qa & \to & p \\ 2pA & \to & qaaA \\ qB & \to & z \\ 1zA & \to & qBA \\ 1zA & \to & fa \\ fa & \to & f \end{array}$$

In the first step, it puts one a on the push-down. Then the same technique as for copying is applied, only for every popped a we put two new ones on the push-down. This doubling of (the number of) as is repeated until the final state f is entered, which simply pops the remaining as. So in $i-1$ steps of copying, we obtain $1+2+4+\cdots+2^{i-1} = 2^i - 1$ terminals. In the step removing A, one more a is produced. Thus the language accepted is $\{a^{2^i} : i \geq 1\}$, which not regular as can be seen with standard pumping arguments, for example.

This suggests that maybe the only interesting cases for Open Problem 6 are the ones of the deep push-down automata in the original definition. Therefore we state it in a more concise form.

Open Problem 7. *For which* i *is* ${}_{deep}\mathbf{PD}_i = \mathbf{REG} = \mathbf{CF}$ *over one-letter alphabet?*

With this we conclude our list of open problems, although it could be expanded further easily.

Acknowledgements

The present author wishes to express his gratefulness to Alexander Meduna for his help. Further, he is thankful for the funding through a post-doctoral fellowship by the Japanese Society for the Promotion of Science under number P07810.

References

[1] M.A. Harrison: *Introduction to Formal Language Theory*. Reading, Mass., 1978.

[2] T. Kasai: *An Infinite Hierarchy between Context-Free and Context-Sensitive Languages*. Journal of Computer and System Sciences 4, 1970, pp. 492–508.

[3] A. Meduna: *Deep Pushdown Automata*. Acta Informatica 42(8-9), 2006, pp. 541–552.