One-Time Nondeterministic Computations

Markus Holzer and Martin Kutrib(✉)

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
{holzer,kutrib}@informatik.uni-giessen.de

Abstract. We introduce the concept of one-time nondeterminism as a new kind of limited nondeterminism for finite state machines and pushdown automata. Roughly speaking, *one-time* nondeterminism means that at the outset the automaton is nondeterministic, but whenever it performs a guess, this guess is fixed for the rest of the computation. We characterize the computational power of one-time nondeterministic finite automata (OTNFAs) and one-time nondeterministic pushdown devices. Moreover, we study the descriptional complexity of these machines. For instance, we show that for an n-state OTNFA with a sole nondeterministic state, that is nondeterministic for only one input symbol, $(n+1)^n$ states are sufficient and necessary in the worst case for an equivalent deterministic finite automaton. In case of pushdown automata, the conversion of a nondeterministic to a one-time nondeterministic as well as the conversion of a one-time nondeterministic to a deterministic one turn out to be non-recursive, that is, the trade-offs in size cannot be bounded by any recursive function.

1 Introduction

The concept of nondeterministic machines was introduced in the seminal paper of Rabin and Scott [13] on finite automata and their decision problems from 1959. Hopcroft [10] writes in his survey "Automata Theory: Its Past and Future" about the nondeterministic finite automaton (NFA) model in the above mentioned paper:

> "It was shown that this model was equivalent to the deterministic one. The fact that two models which were so different gave rise to the same sets, along with the fact that both were equivalent to the regular expression notation, indicated that these models were fundamental. This paper was very influential in shaping automata theory ..."

Nondeterminism turned out to be a very fruitful concept in different areas of computer science like, for example, computability theory, complexity theory, automata theory, formal language theory, etc., to mention only a few. Two of the most prominent problems related to nondeterminism are the P *versus* NP problem (see, for example, [5]) and the LBA problem (see, for example [6]). The former problem is listed as one of the ten Millennium Problems from the

© IFIP International Federation for Information Processing 2017
Published by Springer International Publishing AG 2017. All Rights Reserved
G. Pighizzini and C. Câmpeanu (Eds.): DCFS 2017, LNCS 10316, pp. 177–188, 2017.
DOI: 10.1007/978-3-319-60252-3_14

Clay Mathematics Institute, Cambridge, Massachusetts, USA, which is the sole computer science problem in that list.

Although NFAs are as powerful as deterministic one, as already mentioned above, the former model can offer exponential saving in space compared with deterministic finite automata (DFAs), that is, given some n-state NFA one can always construct a language equivalent DFA with at most 2^n states [13]. This so-called powerset construction turned out to be optimal, in general. That is, the bound on the number of states is tight in the sense that for an arbitrary n there is always some n-state NFA which cannot be simulated by any DFA with less than 2^n states [11,12]. It is worth mentioning that Moore's NFA contains a *sole* nondeterministic state, while Meyer and Fischer's nondeterministic automaton has $n-1$ nondeterministic states. In both NFAs the nondeterministic branching, that is, the maximal number of transitions with the same label leaving a state, is bounded by two, thus by a constant. This can be seen as a first indication that not all NFAs make equal use of nondeterminism. In fact, nondeterminism is a resource and its usage can be accurately quantified. There are several possibilities to do so. For instance, one is to count the number of nondeterministic moves during the computation, while another one depends on the number of the successor states and is called branching and guessing. Results on these quantifications are subsumed under the name *limited nondeterminism* in the literature, see, for example, [1] for a survey.

Here we give a new interpretation of nondeterminism. On the one hand, we are still interested in the power of nondeterminism with respect to computations and conciseness, but on the other hand, its usage should be heavily restricted. The restriction we are introducing is that of *one-time* nondeterminism, which means that at the outset the automaton is nondeterministic, but whenever it performs a guess, this guess is fixed for the rest of the computation. This is a clear change on the semantics of nondeterminism. We will study this new concept for finite automata and pushdown machines. Although one-time nondeterminism does not increase the accepting power of ordinary finite state devices, their conciseness is even greater than that of ordinary nondeterministic finite automata compared to deterministic ones. In general, an n-state one time nondeterministic finite automaton (OTNFA) can be simulated by an $(n+1)^{n^{k \cdot n}}$-state deterministic finite automaton, where k is the size of the input alphabet. In general this bound is over-counted. A slightly better estimate is obtained by using the notion of the *degree of nondeterminism* d for finite state devices, which is defined as the product of all non-zero size successor sets of a state. Formally $d(M) = \prod_{\substack{(q,a)\in Q\times\Sigma \\ |\delta(q,a)|\neq 0}} |\delta(q,a)|$, where Q is the set of states, Σ the input alphabet, and δ the nondeterministic transition function of M. A better upper bound on the above mentioned simulation of an OTNFA M is then $(n+1)^{d(M)}$. This bound is tight in a special case, because we can give an example of an n-state OTNFA M of nondeterministic degree $d(M) = n$ such that $(n+1)^n$ states are sufficient and necessary in the worst case for an equivalent deterministic finite automaton. For pushdown automata the results on the descriptional complexity are even more dramatic. First of all it is shown that one-time nondeterministic

pushdown automata (OTNPDA) accept exactly the union closure of the deterministic context-free languages. This is a well-known language family which properly lies between the deterministic context-free languages and the context free ones. We utilize this characterization and show that the trade-offs between OTNPDAs and deterministic ones, as well as between ordinary nondeterministic pushdown automata and OTNPDAs are non-recursive. That is, the bound between two equivalent machines of different types cannot be bounded by any recursive function. This is quite exceptional because the use of nondeterminism is highly restricted to the bare necessities.

2 Preliminaries

Let Σ^* denote the set of all words over the finite alphabet Σ. The *empty word* is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The *reversal* of a word w is denoted by w^R. For the *length* of w we write $|w|$. For the number of occurrences of a symbol a in w we use the notation $|w|_a$. Set inclusion is denoted by $\subseteq$ and strict set inclusion by $\subset$. We write 2^S for the power set and $|S|$ for the cardinality of a set S.

We recall some notation for descriptional complexity. Following [9] we say that a *descriptional system* $\mathcal{S}$ is a set of finite descriptors such that each $D \in \mathcal{S}$ describes a formal language $L(D)$, and the alphabet $\mathrm{alph}(D)$ over which D represents a language can be deduced from D. The *family of languages represented* (or *described*) by $\mathcal{S}$ is $\mathscr{L}(\mathcal{S}) = \{ L(D) \mid D \in \mathcal{S} \}$. For every language L, the set $\mathcal{S}(L) = \{ D \in \mathcal{S} \mid L(D) = L \}$ is the set of its descriptors in $\mathcal{S}$. A *complexity measure* for a descriptional system $\mathcal{S}$ is a total recursive mapping $c : \mathcal{S} \to \mathbb{N}$.

Finite automata or (deterministic) pushdown automata can be encoded over some fixed alphabet such that their input alphabets can be extracted from the encodings. The sets of these encodings are descriptional systems $\mathcal{S}_1$ and $\mathcal{S}_2$, and $\mathscr{L}(\mathcal{S}_1)$ is the family of regular languages and $\mathscr{L}(\mathcal{S}_2)$ is the family of (deterministic) context-free languages. Examples for complexity measures for finite automata or pushdown automata are the total number of symbols, that is, the *length of the encoding* (length), or, in the former case, the *number of states* and, in the latter case, the product of the number of transition rules and the maximal number of symbols pushed in one step.

Here we only use complexity measures that are recursively related to length. If there is a total recursive function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that, for all $D \in \mathcal{S}$, $\mathsf{length}(D) \leq g(c(D), |\mathrm{alph}(D)|)$, then c is said to be an *s-measure*. If, in addition, for any alphabet Σ, the set of descriptors in $\mathcal{S}$ describing languages over Σ is recursively enumerable in order of increasing size, then c is said to be an *sn-measure*. Clearly, the number of states is an sn-measure for finite automata.

Whenever we consider the relative succinctness of two descriptional systems $\mathcal{S}_1$ and $\mathcal{S}_2$, we assume that the intersection $\mathscr{L}(\mathcal{S}_1) \cap \mathscr{L}(\mathcal{S}_2)$ is non-empty. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be descriptional systems with complexity measures c_1 and c_2, respectively. A total function $f : \mathbb{N} \to \mathbb{N}$ is an *upper bound* for the increase in complexity when changing from a descriptor in $\mathcal{S}_1$ to an equivalent descriptor

in $\mathcal{S}_2$, if for all $D_1 \in \mathcal{S}_1$ with $L(D_1) \in \mathscr{L}(\mathcal{S}_2)$, there exists a $D_2 \in \mathcal{S}_2(L(D_1))$ such that $c_2(D_2) \leq f(c_1(D_1))$. If there is no recursive upper bound, then the *trade-off* for changing from a description in $\mathcal{S}_1$ to an equivalent description in $\mathcal{S}_2$ *is said to be non-recursive*. Non-recursive trade-offs are independent of particular sn-measures.

3 One-Time Nondeterministic Finite Automata

We investigate *one-time nondeterministic* finite automata. The basic idea is that at the outset the automaton is nondeterministic. But whenever it performs a guess, this guess is fixed for the rest of the computation.

A *nondeterministic finite automaton* (NFA) is a system $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, where Q is the finite set of *internal states*, Σ is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and $\delta : Q \times \Sigma \to 2^Q$ is the *transition function*. A *configuration* of a finite automaton M is a tuple (q, w), where $q \in Q$ and $w \in \Sigma^*$. If a is in Σ and w in Σ^*, then we write $(q, aw) \vdash_M (p, w)$ if p is in $\delta(q, a)$. As usual, the reflexive transitive closure of $\vdash_M$ is denoted by $\vdash_M^*$. The subscript M will be dropped from $\vdash_M$ and $\vdash_M^*$ if the meaning is clear. Then the *language accepted* by M is defined as

$$L(M) = \{\, w \in \Sigma^* \mid (q_0, w) \vdash_M^* (p, \lambda) \text{ for some } p \in F \,\}.$$

A state q is *nondeterministic on a letter* a in M, if $|\delta(q, a)| \geq 2$, and state q is *nondeterministic* in M, if it is nondeterministic on some letter $a \in \Sigma$. Moreover, a state q is *reachable* in M if there is an input word w with $(q_0, w) \vdash_M^* (q, \lambda)$. Without loss of generality we assume that any state of a (non)deterministic finite automaton is reachable.

A finite automaton M is *partial deterministic* (partial DFA) if and only if $|\delta(q, a)| \leq 1$, for all $q \in Q$ and $a \in \Sigma$, and the device M is *deterministic* (DFA) if and only if $|\delta(q, a)| = 1$, for every $q \in Q$ and $a \in \Sigma$. In these cases we simply write $\delta(q, a) = q'$ for $\delta(q, a) = \{q'\}$ assuming that the transition function is a (partial) mapping $\delta : Q \times \Sigma \to Q$. Observe, that every partial DFA can be made complete by introducing a non-accepting sink state. So, any DFA is complete, that is, the transition function is total, whereas for partial DFAs and NFAs it is possible that δ maps to the empty set.

Next, the idea of one-time nondeterministic finite automata is formalized as follows: let $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ be an NFA and

$$\begin{aligned}(q_0, w) = (q_0, a_0 w_0) \vdash_M (q_1, w_0) = (q_1, a_1 w_1) \vdash_M \cdots \\ \vdash_M (q_n, w_{n-1}) = (q_n, a_n w_n) \vdash_M (q_{n+1}, w_n) = (q_{n+1}, \lambda),\end{aligned}$$

be a computation of M on input $w = a_0 w_0 \in \Sigma^+$. A computation is *permissible* if and only if $(q_i, a_i) = (q_j, a_j)$ implies $q_{i+1} = q_{j+1}$, for all $0 \leq i < j \leq n$, or the computation is *trivial*, that is, it consists of (q_0, λ) only. Now, M is said to be *one-time nondeterministic* if and only if it may only perform permissible computations. In this case we call M a *one-time nondeterministic finite automaton*

(OTNFA). A word w is permissible acceptable by M if there is a permissible computation that ends in an accepting state of M. The language accepted by an OTNFA M is

$$L_p(M) = \{\, w \in \Sigma^* \mid \text{word } w \text{ is permissible acceptable by } M \,\}.$$

In order to illustrate the definitions we continue with an example.

Example 1. Consider the NFA M with a single nondeterministic state depicted on the left of Fig. 1. It is easy to see that M accepts all words where the next to last letter is an a, that is, the language $(a+b)^*a(a+b)$. When interpreting M as an OTNFA, then only permissible computations are allowed. This means, that for the sole nondeterministic state either the a-self-loop from state 1 to itself or the a-transition leading form state 1 to 2 can be used during the computation, but not both. Thus, we can think of doing a computation either in the finite automaton M' or M'' shown in the middle of or on the right of Fig. 1. Therefore, the language accepted by the OTNFA M is equal to $b^*a(a+b)$, because the automaton in the middle does not accept anything. ■

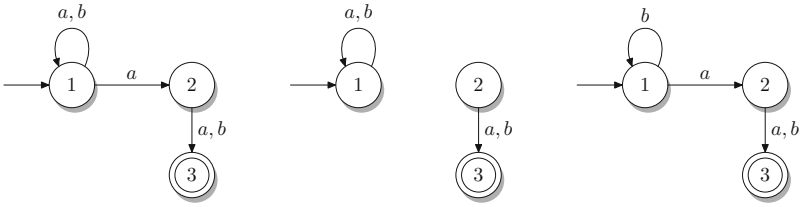

Fig. 1. Left: the NFA M with a sole nondeterministic state accepting the language $(a+b)^*a(a+b)$. Interpreting M as an OTNFA results in accepting $b^*a(a+b)$. Middle: partial DFA M' built from M by choosing the a-self-loop on the state 1 and deleting all other a-transitions leaving state 1. Right: partial DFA M'' built from M by choosing the a-transition from state 1 to 2 and deleting all other a-transitions from state 1.

The following statement is trivial since the permissible computations of an NFA are a subset of all possible computations. The strictness of the inclusion follows by the above given example.

Lemma 2. *Let M be a nondeterministic finite automaton. Then the inclusion $L_p(M) \subseteq L(M)$ holds, and there is a device M such that it is proper.* □

Before we consider the computational power of OTNFAs in more detail, we need some further notation. Let $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ be an NFA. An automaton $M' = \langle Q', \Sigma', \delta', q_0', F\rangle$ is *compatible* with M if and only if (i) $Q' = Q$, (ii) $\Sigma' = \Sigma$, (iii) $q_0' = q_0$, (iv) $F' = F$, and (v) $\delta'(q,a) \subseteq \delta(q,a)$, for every $q \in Q$ and $a \in \Sigma$. If M' is compatible with M, then we write $M' \prec M$. We further define that M' is *non-empty compatible* with M, if and only if M' is compatible

with M and $\delta(q,a) \neq \emptyset$ implies $\delta'(q,a) \neq \emptyset$, for every $q \in Q$ and $a \in \Sigma$. If M' is non-empty compatible with M, then we write $M' \prec_{ne} M$. Note that every automaton is (non-empty) compatible with itself. Obviously, $M' \prec_{ne} M$ implies $M' \prec M$, but the converse implication does not hold in general. Both automata M' and M'' depicted in Fig. 1 are non-empty compatible with M shown in the same figure on the left; for short $M' \prec_{ne} M$ and $M'' \prec_{ne} M$. Now we are ready for the next theorem.

Theorem 3. *Let M be an OTNFA. Then $L_p(M) = \bigcup_{M' \in \mathcal{D}_{ne}(M)} L(M')$, where $\mathcal{D}_{ne}(M) = \{\, M' \mid M' \text{is a partial DFA with} M' \prec_{ne} M \,\}$.*

Proof. The inclusion from left to right is seen as follows: let $M = \langle Q, \Sigma, \delta, q_0, F \rangle$. Consider any word $w \in L_p(M)$. Since w belongs to the language in question, there is a permissible computation of the form (q_0, λ), if $w = \lambda$ or

$$\begin{aligned}(q_0, w) = (q_0, a_0 w_0) &\vdash_M (q_1, w_0) = (q_1, a_1 w_1) \vdash_M \cdots \\ &\vdash_M (q_n, w_{n-1}) = (q_n, a_n w_n) \vdash_M (q_{n+1}, w_n) = (q_{n+1}, \lambda)\end{aligned}$$

with $q_{n+1} \in F$, suitable $a_0, a_1, \ldots, a_n$ in Σ, and words $w_0, w_1, \ldots, w_n$ in Σ^*. It is not hard to see that there is a partial DFA $M' = \langle Q, \Sigma, \delta', q_0, F \rangle$ with $M' \prec_{ne} M$ satisfying $\delta'(q_i, a_i) = q_{i+1}$, for $0 \leq i \leq n$—the non-used transitions during the considered computation are appropriately induced by the OTNFA M. Therefore, the word w is accepted by the partial DFA M', that is, $w \in L(M')$. Thus, w belongs to $\bigcup_{M' \in \mathcal{D}_{ne}(M)} L(M')$, since $M' \in \mathcal{D}_{ne}(M)$. This shows that $L_p(M) \subseteq \bigcup_{M' \in \mathcal{D}_{ne}(M)} L(M')$.

For the converse inclusion we argue as follows: let M' be any partial DFA from $\mathcal{D}_{ne}(M)$. Consider a word $w \in L(M')$. As above, there is a computation of the form (q_0, λ), if $w = \lambda$ or

$$\begin{aligned}(q_0, w) = (q_0, a_0 w_0) &\vdash_M (q_1, w_0) = (q_1, a_1 w_1) \vdash_M \cdots \\ &\vdash_M (q_n, w_{n-1}) = (q_n, a_n w_n) \vdash_M (q_{n+1}, w_n) = (q_{n+1}, \lambda)\end{aligned}$$

with $q_{n+1} \in F$, suitable letters $a_0, a_1, \ldots, a_n$ in Σ, and words $w_0, w_1, \ldots, w_n$ in Σ^*. Since this is a deterministic computation one can interpret it as a *permissible* computation of M, since M can simulate every step of the partial DFA M'. Therefore we conclude that $w \in L_p(M)$, hence $L_p(M) \supseteq L(M')$. Note that the given argumentation holds for every $M' \in \mathcal{D}_{ne}(M)$ and therefore $L_p(M) \supseteq \bigcup_{M' \in \mathcal{D}_{ne}(M)} L(M')$. This proves the stated claim. □

Theorem 3 shows that OTNFAs accept only regular languages, because they are closed under finite union. Although OTNFAs do not improve the computational power of ordinary finite automata, the question on the descriptional complexity of these devices arises. The statement of Theorem 3 give us some hint, that the relative succinctness compared to ordinary finite state devices may be enormous, since the union on the right-hand side of $L_p(M) = \bigcup_{M' \in \mathcal{D}_{ne}(M)} L(M')$ runs over a large number of different M' from $\mathcal{D}_{ne}(M)$. In fact, our intuition is supported by the next result, which states a very large upper bound.

Theorem 4. *Let M be an n-state OTNFA with a k-letter input alphabet. Then $(n+1)^{n^{k \cdot n}}$ states are sufficient for a DFA to accept the language $L_p(M)$.*

Proof. Let $M = \langle Q, \Sigma, \delta, q_0, F \rangle$. Then $L_p(M) = \bigcup_{M' \in \mathcal{D}_{ne}(M)} L(M')$ by Theorem 3, where $\mathcal{D}_{ne}(M) = \{ M' \mid M' \text{ is a partial DFA with } M' \prec_{ne} M \}$. In order to accept the union on the right-hand side by a DFA, we apply the standard cross-product construction [14]. To this end, we complete the involved partial automata from $\mathcal{D}_{ne}(M)$, which results in ordinary DFAs with at most $n+1$ states. Then the upper bound on the number of states for any DFA accepting $L_p(M)$ is $(n+1)^{|\mathcal{D}_{ne}(M)|}$. Thus, it remains to obtain an upper bound on the size of $\mathcal{D}_{ne}(M)$.

An automaton M' in $\mathcal{D}_{ne}(M)$ is constructed from M by cycling trough all states q and letters a and distinguishing the following cases: (i) if $\delta(q,a) = \emptyset$, then $\delta'(q,a) = \emptyset$, (ii) if $\delta(q,a) = \{p\}$, then $\delta'(q,a) = \{p\}$, and (iii) if $\delta(q,a) = \{p_1, p_2, \ldots, p_k\}$, for $k \geq 2$, then $\delta'(q,a) = \{p_i\}$, for some i with $1 \leq i \leq k$. Since $|\delta(q,a)| \leq n$, for every state q and letter a, we conclude that the size of $\mathcal{D}_{ne}(M)$ is bounded from above by $n^{k \cdot n}$. Hence we get the stated upper bound of $(n+1)^{n^{k \cdot n}}$ states for a DFA accepting the language $L_p(M)$. □

It turns out that the upper bound on the number of states of a DFA accepting $L_p(M)$, for a OTNFA M, is more accurate the better we can determine the size of $\mathcal{D}_{ne}(M)$. A greater precision is obtained by defining the *nondeterministic degree* $d(M)$ of an OTNFA M with state set Q and input alphabet Σ as $d(M) = \prod_{\substack{(q,a) \in Q \times \Sigma \\ |\delta(q,a)| \neq 0}} |\delta(q,a)|$. Then it is easy to see that the following lemma holds.

Lemma 5. *Let M be an OTNFA. Then $|\mathcal{D}_{ne}(M)| = d(M)$.* □

Hence we can reformulate the upper bound given in Theorem 4 as follows:

Corollary 6. *Let M be an OTNFA with nondeterministic degree $d(M)$. Then any DFA that accepts the language $L_p(M)$ needs at most $(n+1)^{d(n)}$ states.* □

In the remainder of this section we obtain that the bound stated in the previous corollary can be reached already for a OTNFA with a sole nondeterministic state, that is nondeterministic for only one input symbol.

Theorem 7. *There is a n-state OTNFA M with a sole nondeterministic state, that is nondeterministic only for one input symbol, and has nondeterministic degree n, such that $(n+1)^n$ states are sufficient and necessary in the worst case for a DFA to accept the language $L_p(M)$.*

Proof. The upper bound of $(n+1)^{d(n)}$ states for a DFA accepting the language $L_p(M)$ follows from Corollary 6. For the lower bound we argue as follows: consider the OTNFA whose transition function δ on the letters a, b, c, and d is depicted in Fig. 2.

By the $\prec_{ne}$-relation the OTNFA M gives rise to several partial DFAs M_j, for $1 \leq j \leq n$, where $M_j = \langle \{1, 2, \ldots, n\}, \{a, b, c, d\}, \delta_j, 1, \{n\} \rangle$ and the transition

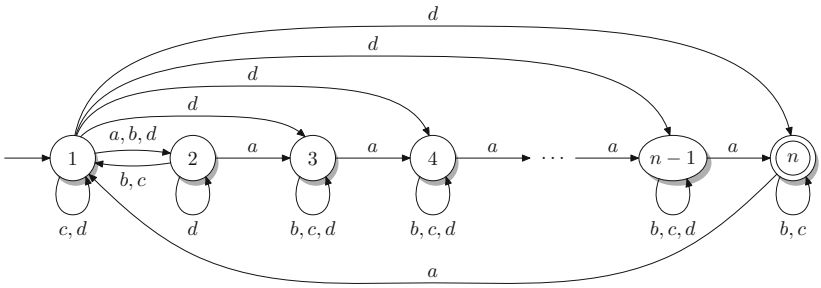

Fig. 2. The transition function δ of the OTNFA M.

function δ_j is equal to δ on the letters a, b, and c, and for the letter d we have $\delta_j(i,d) = \{j\}$, if $i = 1$, $\delta_j(i,d) = \{i\}$, if $2 \leq i < n$, and $\delta_j(i,d)$ is undefined otherwise. These partial DFAs $M_1, M_2, \ldots, M_n$ are the basic building blocks in the cross-product construction for the (ordinary) DFA M' accepting the language $L_p(M)$. Define the DFA $M' = \langle Q', \{a,b,c,d\}, \delta', q_0', F' \rangle$, where the set of states is equal to $Q' = (\{1,2,\ldots,n\} \cup \{-\})^n$, the initial state is $q_0' = (1,1,\ldots,1)$, and $F' = \{\, f \in Q' \mid f[i] = n, \text{for some } 1 \leq i \leq n \,\}$, where $f[i]$ refers to the ith component of $f \in (\{1,2,\ldots,n\} \cup \{-\})^n$. The transition function is given by $\delta'(f,a) = (\delta_1(f[1],a), \delta_2(f[2],a), \ldots, \delta_n(f[n],a))$, for every $f \in Q'$ and $a \in \Sigma$. Here we assume that whenever some $\delta_i(f[i],a)$ is *undefined*, then the component is set to $-$. In order to prove our statement we need to show that the DFA M' is minimal. The proof that every state in M' is reachable and defines a distinct equivalence class utilizes some results from semigroup theory since one can identify the states of M' with partial mappings from $[n]$ to $[n]$. □

4 One-Time Nondeterministic Pushdown Automata

Now we turn to generalize the definitions of OTNFAs to pushdown automata. Nondeterministic pushdown automata are well known for capturing the context-free languages.

Let Σ be an alphabet. For convenience, we use Σ_λ for $\Sigma \cup \{\lambda\}$. A *nondeterministic pushdown automaton* (NPDA) is a system $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F \rangle$, where Q is a finite set of *internal states*, Σ is the finite set of *input symbols*, Γ is a finite set of *pushdown symbols*, δ is a mapping from $Q \times \Sigma_\lambda \times \Gamma$ to finite subsets of $Q \times \Gamma^*$ called the *transition function*, $q_0 \in Q$ is the *initial state*, $\bot \in \Gamma$ is the so-called *bottom-of-pushdown symbol*, which initially appears on the pushdown store, and $F \subseteq Q$ is the set of *accepting states*.

A *configuration* of a pushdown automaton is a triple (q, w, γ), where q is the current state, w the unread part of the input, and γ the current content of the pushdown store, the leftmost symbol of γ being the top symbol. On input w the initial configuration is defined to be $(q_0, w, \bot)$. If p, q are in Q, a is in Σ_λ, w is in Σ^*, γ and β are in Γ^*, and Z is in Γ, then we write $(q, aw, Z\gamma) \vdash_M (p, w, \beta\gamma)$,

if the pair (p, β) is in $\delta(q, a, Z)$. In order to simplify matters, we require that during any computation the bottom-of-pushdown symbol appears only at the bottom of the pushdown store. Formally, we require that if (p, β) is in $\delta(q, a, Z)$, then either β does not contain $\bot$ or $\beta = \beta'\bot$, where β' does not contain $\bot$, and $Z = \bot$. As usual, the reflexive transitive closure of $\vdash_M$ is denoted by $\vdash^*_M$. The subscript M will be dropped whenever the meaning remains clear. The *language accepted* by M with accepting states is

$$L(M) = \{\, w \in \Sigma^* \mid (q_0, w, \bot) \vdash^* (q, \lambda, \gamma), \text{ for some } q \in F \text{ and } \gamma \in \Gamma^* \,\}.$$

An NPDA is a *deterministic pushdown automaton* (DPDA), if there is at most one choice of action for any possible configuration. In particular, there must never be a choice of using an input symbol or of using λ input. Formally, a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F\rangle$ is *deterministic* if (i) $\delta(q, a, Z)$ contains at most one element, for all a in Σ_λ, q in Q, and Z in Γ, and (ii) for all q in Q and Z in Γ: if $\delta(q, \lambda, Z)$ is not empty, then $\delta(q, a, Z)$ is empty for all a in Σ.

Now we turn to *one-time nondeterministic* pushdown automata (OTNPDA). As before, whenever such an automaton performs a guess, this guess is fixed for the rest of the computation. Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F\rangle$ be an NPDA and, for $q_i \in Q$, $a_i \in \Sigma_\lambda$, $0 \le i \le n$, and $Z_i \in \Gamma$, $\beta_i \in \Gamma^*$, $1 \le i \le n$,

$$\begin{aligned}(q_0, a_0w_0, \bot) \vdash_M (q_1, w_0, \beta_1) = (q_1, a_1w_1, Z_1\gamma_1) \vdash_M (q_2, w_1, \beta_2\gamma_1) = \\ (q_2, a_2w_2, Z_2\gamma_2) \vdash^*_M (q_n, w_{n-1}, \beta_n\gamma_{n-1}) = (q_n, a_nw_n, Z_n\gamma_n)\end{aligned}$$

be a computation of M on input $a_0w_0 \in \Sigma^+$. The computation is defined to be *permissible* if and only if, (i) the equality $(q_i, Z_i) = (q_j, Z_j)$ implies either $a_i = a_j = \lambda$ or $a_i \ne \lambda$ and $a_j \ne \lambda$, and (ii) the equality $(q_i, a_i, Z_i) = (q_j, a_j, Z_j)$ implies $q_{i+1} = q_{j+1}$ and $\beta_{i+1} = \beta_{j+1}$, for all $0 \le i < j \le n-1$. A word w is *permissible acceptable* by M if on input w there is an accepting permissible computation. The language accepted by an OTNPDA M is

$$L_p(M) = \{\, w \in \Sigma^* \mid \text{word } w \text{ is permissible acceptable by } M \,\}.$$

In order to derive the relationships of OTNPDAs, NPDAs, and DPDAs, we generalize the characterization of OTNFAs to pushdown automata. The notation as well as Theorem 3 is straightforwardly adapted. So, we obtain:

Theorem 8. *Let M be a OTNPDA. Then $L_p(M) = \bigcup_{M' \in \mathcal{D}(M)} L(M')$, where $\mathcal{D}_{ne}(M) = \{\, M' \mid M' \text{ is a DPDA with } M' \prec_{ne} M \,\}$.*

So, the family of languages accepted by OTNPDAs is included in the (finite) union closure of the deterministic context-free languages. Conversely, the union of a finite number of languages accepted by DPDAs is accepted by an NPDA whose first transition is a nondeterministic choice of which of the DPDAs is to be simulated. Subsequently, the NPDA simulates the chosen DPDA. In this way, the nondeterministic situation at the beginning is never reached again and,

thus, the NPDA is in fact a OTNPDA. We conclude that the union closure of the deterministic context-free languages is included in the family of languages accepted by OTNPDAs.

Corollary 9. *The family of languages accepted by OTNPDAs and the union closure of the deterministic context-free languages are identical.* □

This characterization together with known results shows that the computational capacity of OTNPDAs is strictly in between the NPDAs and DPDAs.

Theorem 10. $\mathscr{L}(DPDA) \subset \mathscr{L}(OTNPDA) \subset \mathscr{L}(NPDA)$.

For establishing non-recursive trade-offs the following general result is useful that is a slightly generalized and unified form of a result of Hartmanis [4].

Theorem 11 [9]**.** *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two descriptional systems for recursive languages such that any descriptor D in $\mathcal{S}_1$ and $\mathcal{S}_2$ can effectively be converted into a Turing machine that decides $L(D)$, and let c_1 be a measure for $\mathcal{S}_1$ and c_2 be an sn-measure for $\mathcal{S}_2$. If there exists a descriptional system $\mathcal{S}_3$ and a property P that is not semi-decidable for descriptors from $\mathcal{S}_3$, such that, given an arbitrary $D_3 \in \mathcal{S}_3$, (i) there exists an effective procedure to construct a descriptor D_1 in $\mathcal{S}_1$, and (ii) D_1 has an equivalent descriptor in $\mathcal{S}_2$ if and only if D_3 does not have property P, then the trade-off between S_1 and S_2 is non-recursive.*

By measuring the amount of ambiguity and nondeterminism in pushdown automata in [7,8] infinite hierarchies in between the deterministic and nondeterministic context-free languages are obtained. Intuitively, the corresponding language families are close together. Nevertheless, there are non-recursive trade-offs between the levels of the hierarchies.

In the following we show non-recursive trade-offs between nondeterministic and OTNPDAs automata as well as between OTNPDAs deterministic pushdown automata by reduction from the non-halting problem for Turing machines on empty tape. To this end, histories of Turing machine computations are encoded into strings, a technique introduced in [3]. It suffices to consider deterministic Turing machines with one single tape and one single read-write head. Without loss of generality and for technical reasons, we assume that the Turing machines cannot print blanks, can halt only after an odd number of moves, accept by halting, and visit an infinite number of squares if they do not halt.

Let Q be the state set of some Turing machine M, where q_0 is the initial state, $T \cap Q = \emptyset$ is the tape alphabet containing the blank symbol, and $\Sigma \subset T$ is the input alphabet. Then a configuration of M can be written as a word of the form T^*QT^* such that $t_1 t_2 \cdots t_i q t_{i+1} \cdots t_n$ is used to express that M is in state q, scanning tape symbol t_{i+1}, and t_1, t_2 to t_n is the support of the tape inscription. Dependent on the Turing machine M we define the following two languages. Let $a, b, \$, \# \notin T \cup Q$, $n \geq 0$, and $w_i \in T^*QT^*$, $0 \leq i \leq 2n+1$ are configurations of M. Then language $L_{M,0}$ contains all words of the form $\$w_0\$w_1^R\$w_2\$w_3^R\$ \cdots \$w_{2n}\#w_{2n+1}^R\#a$, where w_0 is an initial configuration with empty tape of the form $w_0 = q_0$ and w_{2i} is the successor configuration of w_{2i-1}, $1 \leq i \leq n$, and language $L_{M,1}$ contains all words of the form

$\$w_0\$w_1^R\$w_2\$w_3^R\$\cdots\$w_{2n}\#w_{2n+1}^R\#b$, where w_{2i+1} is the successor configuration of w_{2i}, $0 \leq i \leq n$. As an immediate observation we obtain the next corollary.

Corollary 12. *For any deterministic Turing machine M, both languages $L_{M,0}$ and $L_{M,1}$ are deterministic context-free languages, such that their deterministic pushdown automata can effectively be constructed from M.* □

In order to apply Theorem 11, we use the family of deterministic one-tape Turing machines as descriptional system $\mathcal{S}_3$, and for the property P we take the property of *not halting on empty tape*. Recall that this property is indeed not semi-decidable for deterministic one-tape Turing machines. Next, given an arbitrary deterministic one-tape Turing machine M, that is, a descriptor $D_3 \in \mathcal{S}_3$, we must construct a nondeterministic pushdown automaton, that is, a descriptor D_1 in $\mathcal{S}_1$, that has an equivalent one-time nondeterministic pushdown automaton, that is, a descriptor in $\mathcal{S}_2$, if and only if M halts on empty tape.

So, given a deterministic one-tape Turing machine M, we use a new symbol c and define $L_M = ((L_{M,0} \cup L_{M,1})c)^*$. By Corollary 12 and the effective closures of the context-free languages under union and marked Kleene star, we derive that L_M is a context-free language, such that its nondeterministic pushdown automaton D_1 can effectively be constructed from M. It remains to be shown that L_M is accepted by a one-time nondeterministic pushdown automaton if and only if M halts on empty tape. The next lemma shows that L_M is accepted even by a deterministic pushdown automaton if M halts on empty tape.

Lemma 13. *Let M be a deterministic Turing machine that* halts *on empty tape. Then language L_M is accepted by some DPDA.*

In order to disprove that L_M is accepted by a OTNPDA if M does not halt on empty tape Ogden's lemma for deterministic context-free languages is used (see, for example, [2]). Then one can show the following result.

Lemma 14. *Let M be a deterministic Turing machine that* does not halt *on empty tape. Then language L_M is not accepted by any OTNPDA.*

So, we have shown the following theorem.

Theorem 15. *The trade-off between nondeterministic pushdown automata and one-time nondeterministic pushdown automata is non-recursive.* □

The proof of the second non-recursive trade-off between one-time nondeterministic pushdown automata and deterministic pushdown automata follows along the lines of the first non-recursive trade-off. Again, we apply Theorem 11 but now with a simplification of the language L_M. More precisely, given a deterministic one-tape Turing machine M, we define $L'_M = L_{M,0} \cup L_{M,1}$. By Corollary 12, both languages $L_{M,0}$ and $L_{M,1}$ are deterministic context-free languages, such that their deterministic pushdown automata can effectively be constructed from M. So, L'_M is accepted by a one-time nondeterministic pushdown automaton that can effectively be constructed from M. As before, it remains to

be shown that L'_M is accepted by a deterministic pushdown automaton if and only if M halts on empty tape. If M halts on empty tape, Lemma 13 says that L_M is accepted by some DPDA. Since the deterministic context-free languages are closed under intersection with regular sets, this implies that L'_M is accepted by some DPDA either.

Lemma 16. *Let M be a deterministic Turing machine that* does not halt *on empty tape. Then language L'_M is not accepted by any DPDA.*

Thus, we have also shown the following non-recursive trade-off.

Theorem 17. *The trade-off between one-time nondeterministic pushdown automata and deterministic pushdown automata is non-recursive.* □

References

1. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. Univ. Comput. Sci. **8**(2), 193–234 (2002)
2. Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Reading (1978)
3. Hartmanis, J.: Context-free languages and turing machine computations. Proc. Symp. Appl. Math. **19**, 42–51 (1967)
4. Hartmanis, J.: On Gödel speed-up and succinctness of language representations. Theoret. Comput. Sci. **26**, 335–342 (1983)
5. Hartmanis, J.: Gödel, von Neumann and the P=? NP problem. Bull. EATCS **38**, 101–107 (1989)
6. Hartmanis, J., Hunt III., H.B.: The LBA problem and its importance in the theory of computing. In: Complexity of Computing, SIAM AMS Proceedings, vol. 7, pp. 1–26 (1974)
7. Herzog, C.: Pushdown automata with bounded nondeterminism and bounded ambiguity. Theoret. Comput. Sci. **181**, 141–157 (1997)
8. Herzog, C.: Die Rolle des Nichtdeterminismus in kontextfreien Sprachen. Doctoral dissertation, Universität Frankfurt (1999). (in German)
9. Holzer, M., Kutrib, M.: Descriptional complexity - an introductory survey. In: Scientific Applications of Language Methods, pp. 1–58. Imperial College Press (2010)
10. Hopcroft, J.: Automata theory: its past and future. In: Yu, S. (ed.) A Half-Century of Automata Theory, pp. 37–47. World Scientific (2001)
11. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proceedings of the 12th Annual Symposium on Switching and Automata Theory, pp. 188–191. IEEE Computer Society Press (1971)
12. Moore, F.R.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans. Comput. **20**, 1211–1219 (1971)
13. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3**, 114–125 (1959)
14. Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages. Theoret. Comput. Sci. **125**, 315–328 (1994)