Acta Informatica (2021) 58:335–356
https://doi.org/10.1007/s00236-021-00398-7

ORIGINAL ARTICLE

The descriptional power of queue automata of constant length

Sebastian Jakobi[1] · Katja Meckel[1] · Carlo Mereghetti[2] · Beatrice Palano[3]

Received: 1 February 2020 / Accepted: 9 April 2021

Abstract
We consider the notion of a *constant length queue automaton*—i.e., a traditional queue automaton with a built-in constant limit on the length of its queue—as a formalism for representing regular languages. We show that the descriptional power of constant length queue automata greatly outperforms that of traditional finite state automata, of constant height pushdown automata, and of straight line programs for regular expressions, by providing optimal exponential and double-exponential size gaps. Moreover, we prove that constant height pushdown automata can be simulated by constant length queue automata paying only by a linear size increase, and that removing nondeterminism in constant length queue automata requires an optimal exponential size blow-up, against the optimal double-exponential cost for determinizing constant height pushdown automata. Finally, we investigate the size cost of implementing Boolean language operations on deterministic and nondeterministic constant length queue automata.

Mathematics Subject Classification 68Q10 · 68Q45

A very preliminary version of this work was presented at the *15th International Conference on Descriptional Complexity of Formal Systems (DCFS), London, Ontario, Canada, July 22-25, 2013, and it is published in* [21].

✉ Carlo Mereghetti
carlo.mereghetti@unimi.it

Sebastian Jakobi
sebastian.jakobi@informatik.unigiessen.de

Katja Meckel
katja.meckel@informatik.uni-giessen.de

Beatrice Palano
palano@di.unimi.it

[1] Giessen, Germany

[2] Dipartimento di Fisica "Aldo Pontremoli", Università degli Studi di Milano, via Celoria 16, 20133 Milan, Italy

[3] Dipartimento di Informatica "Giovanni Degli Antoni", Università degli Studi di Milano, via Celoria 18, 20133 Milan, Italy

1 Introduction

It is well known that the *computational power* of computing devices can be tuned by restricting the way they access memory. To catch a glimpse of this phenomenon, one may start from the traditional model of a one-way Turing machine equipped with a potentially unbounded and freely accessible working tape representing memory. If we impose a LIFO usage of the working tape, still keeping unboundedness, then we obtain a *pushdown automaton*, whose computational power is strictly lower than that of a one-way Turing machine. Instead, by imposing a FIFO memory access policy, we get *a queue automaton*, whose computational power gets back to that of a one-way Turing machine. However, for all these (and other) models, by fixing a *constant*[1]—i.e., not depending on the input length—bound on the amount of available memory, we have that their computational power boils down to that of finite state automata, regardless of the memory access policy in use.

For *constant memory machines*, it is then worth investigating how the memory access policy affects their *descriptional power*, that is, their capability of succinctly representing regular languages. (We refer the reader to, e.g., [19] for a thoughtful survey on descriptional complexity theory.)

This line of research is settled in [17], where the notion of a *constant height pushdown automaton* is introduced and studied from a descriptional complexity perspective. Roughly speaking, a constant height pushdown automaton is a traditional pushdown automaton with a built-in constant limit on the height of the pushdown. Optimal exponential and double-exponential gaps are proved, between the size of constant height deterministic and nondeterministic pushdown automata (DPDAs and NPDAs, respectively) and the size of equivalent deterministic and nondeterministic finite state automata (DFAs and NFAs, respectively) and classical regular expressions. Moreover, the notion of a straight line program for regular expressions (SLP, see Sect. 2) is also introduced in [17], as a formalism equivalent to a constant height NPDA from a size point of view. In [5], the fundamental problem of removing nondeterminism in constant height NPDAs is tackled, and a double-exponential size blow-up for determinization is emphasized. Finally, the size cost of boolean operations on constant height DPDAs and NPDAs is analyzed in [3,4,6].

In this paper, we investigate the descriptional advantages of replacing the pushdown with a *queue* storage of fixed size, by considering the notion of a *constant length queue automaton*. Basically, this device is a traditional queue automaton (see, e.g., [2,14]), in which the length of the queue cannot grow beyond a fixed constant limit.

As for constant height pushdown automata, in Sect. 3 we single out optimal exponential and double-exponential gaps between the size of constant length deterministic and nondeterministic queue automata (DQAs and NQAs, respectively) and the size of equivalent DFAs and NFAs. In addition, differently from constant height pushdown automata, in Sect. 4 we prove that a queue storage enables a size-efficient removal of nondeterminism. Precisely, we show that NFAs can be simulated by constant length DQAs paying by only a *linear* size increase. This, in turn, leads us to prove that the optimal size cost of removing nondeterminism in constant length NQAs is only *exponential*, in sharp contrast with the optimal double-exponential size blow-up above pointed out for determinizing constant height NPDAs.

The higher descriptional power of a queue vs. a pushdown storage for constant memory machines is also emphasized in Sect. 5, where we show that constant height NPDAs (resp., DPDAs) can be simulated by constant length NQAs (resp., DQAs), paying by only a *linear* size increase. On the other hand, the opposite simulation features an optimal exponential size

[1] Actually, even $o(\log n)$ or $o(\log \log n)$ space bounds (see, e.g., [29]).

cost. This is witnessed by proving, in Sect. 6, that constant length DQAs can be exponentially smaller than equivalent SLPs, this gap being optimal. In Sect. 7, a complete overview on the optimal simulation size costs between finite state automata, constant height pushdown automata, constant length queue automata, and straight line programs is portrayed for reader's convenience. In Sect. 8, as typically done in the literature for formalisms defining regular languages (see, e.g., [3,4,6,16,19,22–24]), the problem of establishing the size cost of implementing boolean language operations on constant length DQAs and NQAs is tackled. As a useful tool in this investigation, analogously to what is done for constant height pushdown automata [3–6,17], a normal form for constant length queue automata is defined, where at most one symbol is enqueued on each move. The size cost of converting constant length queue automata into their normal form is analyzed. Finally, in Sect. 9, we sum up obtained results and provide possible developments of our work.

2 Preliminaries: adding memory to finite state automata

We assume the reader is familiar with basics in formal language theory, and we refer to, e.g., [18,20] for an extensive presentation of the topic. The set of all words (including the empty word ε) on a finite alphabet Σ is denoted by Σ^*. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, and we let Σ^i be the set of words on Σ of length $i \geq 0$ (with $\Sigma^0 = \{\varepsilon\}$). Moreover, we let $\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i$. A language L on Σ is any subset $L \subseteq \Sigma^*$. The complement of L is denoted by L^c.

Finite state automata. A *nondeterministic finite state automaton* (NFA, see, e.g., [18,20]) is formally defined as a 5-tuple $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is the finite set of states, Σ the finite input alphabet, $q_0 \in Q$ the initial state, $F \subseteq Q$ the set of final, or accepting, states, and δ is the transition function mapping $Q \times (\Sigma \cup \{\varepsilon\})$ to finite subsets of Q. The computation of A on an input string $x = \sigma_1\sigma_2 \cdots \sigma_n \in \Sigma^*$ begins in the initial state q_0 by scanning the first input symbol σ_1. Next, the rules established by the transition function δ are subsequently applied. An application of δ is called *move*. An input string is *accepted*, if there exists a computation beginning in the state q_0 and ending in some final state $q \in F$ after reading the entire input. If such a computation does not exist, then the input string is rejected. The set of all inputs accepted by A is denoted by $L(A)$ and called the accepted language. The automaton A is *deterministic* (DFA), if there are no ε-transitions in δ and, for every $q \in Q$ and $a \in \Sigma$, we have $|\delta(q, a)| \leq 1$.

The other two computational models we shall be dealing with can be obtained by equipping finite state automata with some auxiliary memory storage. Depending on whether such memory is used in a LIFO- or FIFO-mode, we have pushdown or queue automata, respectively. In particular, we will be interested in the case in which the auxiliary memory storage has a *fixed constant size* (not depending on input length).

Constant height pushdown automata. A *nondeterministic pushdown automaton* (NPDA, see, e.g., [18,20]) is formally defined as a 7-tuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F \rangle$, where Q, Σ, q_0 and F are defined as for NFAs, Γ is the pushdown alphabet, $\bot \in \Gamma$ is the initial symbol in the pushdown store, and the transition function δ maps $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$. At any time, the pushdown content may be represented by a string where the leftmost symbol is the *top* of the pushdown, while the rightmost is its *bottom*. Let $\delta(q, \sigma, X) \ni (p, \gamma)$. Then A, being in the state q, reading the input symbol σ and the pushdown symbol X on the top of the pushdown, can reach the state p, replace X by γ, and finally advance input scanning to the next input symbol only if $\sigma \neq \varepsilon$. An input string is *accepted*,

if there exists a computation beginning in the state q_0 with $\perp$ in the pushdown, and ending in some final state $q \in F$ after reading the entire input. The set of all inputs accepted by A is denoted by $L(A)$. The automaton A is *deterministic* (DPDA), if for any $q \in Q, \sigma \in \Sigma \cup \{\varepsilon\}$ and $X \in \Gamma$, we have $|\delta(q, \sigma, X)| \leq 1$, and if $\delta(q, \varepsilon, X)$ is defined then $|\delta(q, a, X)| = 0$ for any $a \in \Sigma$.

A *constant height* NPDA [17] is obtained from a traditional NPDA by imposing that the pushdown store can never contain more than h pushdown symbols, for a fixed constant $h \geq 0$ not depending on input length. By definition, any attempt to store more than h symbols in the pushdown results in rejection. This constant memory bounded device is formally specified by an 8-tuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F, h \rangle$, where $h \geq 0$ is the built-in pushdown height, while all other components are defined as above.

Constant length queue automata. A *nondeterministic queue automaton* (NQA, see, e.g., [2,14]) is formally defined as a 7-tuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F \rangle$, where Q, Σ, q_0, F are defined as for NFAs, Γ is the queue alphabet, $\vdash \in \Gamma$ is the initial symbol in the queue store, and the transition function δ maps $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \{D, K\} \times \Gamma^*$. At any time, the queue content may be represented by a string where the leftmost symbol is the *head* of the queue, while the rightmost is its *tail*. Let $\delta(q, \sigma, X) \ni (p, \chi, \omega)$. Then A, being in the state q, reading σ on the input and X as the head of the queue, can reach the state p, delete (resp., keep) X if $\chi = D$ (resp., $\chi = K$), enqueue ω (i.e., append ω after the tail), and finally advance input scanning to the next input symbol only if $\sigma \neq \varepsilon$. So, a transition with parameter D (resp., K) means that we delete (resp., keep) the symbol at the head of the queue. An input string is *accepted*, if there exists a computation beginning in the state q_0 with $\vdash$ in the queue, and ending in some final state $q \in F$ after reading the entire input. The set of all inputs accepted by A is denoted by $L(A)$. The automaton A is *deterministic* (DQA), if for any $q \in Q, \sigma \in \Sigma \cup \{\varepsilon\}$ and $X \in \Gamma$, we have $|\delta(q, \sigma, X)| \leq 1$, and if $\delta(q, \varepsilon, X)$ is defined then $|\delta(q, a, X)| = 0$ for any $a \in \Sigma$.

A quick comment on our definition of a queue automaton is in order. In the definition of several automata models, two special types of moves, namely *stationary moves* and *ε-moves*, are sometimes allowed. Basically, a stationary moves takes place on well-defined "configurations" established in the transition function where a particular state and a particular input symbol must be scanned. Acting such a move may possibly modify the state of the automaton, some auxiliary storage (if any), while the input head has to stay put. On the other hand, an ε-move is defined on a particular state without the need of reading a particular input symbol. After an ε-move, the state and some storage (if any) content may be changed, while the input head is allowed to move or stay put.

In our definition of a queue automaton, we allow ε-moves but not stationary moves on the input, this latter feature being usually assumed (see, e.g., [2,14]). Our choice is motivated by guaranteeing direct and fair comparisons between queue automata and pushdown automata where stationary moves are never considered. However, it is not hard to see that queue automata with ε-moves and queue automata with stationary moves are descriptionally equivalent.

A *constant length* NQA is obtained from a traditional NQA by imposing that the queue can never contain more than h symbols, for a fixed constant $h \geq 0$ not depending on input length. Any attempt to store more than h symbols in the queue results in rejection. This constant

memory bounded device is formally specified by an 8-tuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$, where $h \geq 0$ is the built-in queue length, while all other components are defined as above.

Some common notions and observations. Throughout the rest of the paper, we will say that two automata A and A' are *equivalent* whenever $L(A) = L(A')$ holds true. Moreover, for the sake of conciseness and when no possible confusion arises, we will be using the designation *constant memory automaton* to denote either a constant height NPDA or a constant length NQA. We observe that a constant memory automaton can be replaced by a corresponding "traditional"—i.e., without a built-in limit h on its memory storage—automaton by storing in its finite control states a counter recording permitted memory amounts (i.e., a number ranging within $\{1, \ldots, h\}$). This increases the number of states by the multiplicative factor h (see, e.g., Lemma 3, where this technique is used in the proof). Indeed, notice that, for $h = 0$, the definition of a constant memory automaton exactly coincides with that of a finite state automaton.

Concerning acceptance mode, our constant memory automata are defined to accept by final states. However, in case of *nondeterministic* devices, at the cost of one more state it is always possible to accept by a *single final state* and with *empty memory* at once. This result is proved, e.g., in [17] for constant height NPDA; for constant length NQA, the same can be obtained with a similar approach.

For a constant memory automaton, a fair *size measure* (see, e.g., [3,17]) should take into account all the components the device consists of, namely: (i) the number of finite control states, (ii) the size of the memory alphabet, and (iii) the built-in memory limit. So, we adopt the following

Definition 1 The *size* of a constant memory automaton with state set Q, memory alphabet Γ, and memory limit h, is specified by measuring $|Q|$, $|\Gamma|$, and h.

We observe that this definition immediately implies that the size of an NFA, for which clearly $|\Gamma| = 0 = h$ holds true, is completely determined by the number of its states.

Straight line programs for regular expressions. A *regular expression* over a given alphabet Σ is inductively defined as:

(i) $\emptyset$, ε, or a for any symbol $a \in \Sigma$,
(ii) $(r_1 + r_2)$, $(r_1 \cdot r_2)$, or r_1^*, if r_1 and r_2 are regular expressions.

The language represented by a given regular expression is defined in the usual way (see, e.g., [18,20]). With a slight abuse of terminology, we will often identify a regular expression with the language it represents.

A convenient and concise way of specifying regular expressions is provided by straight line programs. Given a set of variables $X = \{x_1, \ldots, x_\ell\}$, a *straight line program for regular expressions* (SLP, see [17]) on Σ is a finite sequence of instructions

$$\begin{aligned} P \equiv\ & \mathrm{instr}_1; \\ & \vdots \\ & \mathrm{instr}_i; \\ & \vdots \\ & \mathrm{instr}_\ell; \end{aligned}$$

where the ith instruction instr_i has one of the following forms:

(i) $x_i := \emptyset$, $x_i := \varepsilon$, or $x_i := a$ for any symbol $a \in \Sigma$,

(ii) $x_i := x_j + x_k$, $x_i := x_j \cdot x_k$, or $x_i := x_j^*$, for $1 \leq j, k < i$.

The program P expands to the regular expression in x_ℓ (output variable), obtained by nested macro-expansion of the variables $x_1, \ldots, x_{\ell-1}$, using the right parts of their instructions. Note that point (ii) in the definition of instruction form imposes a loopless structure to SLPs, naturally leading to their digraph representation analogous to that of boolean circuits (see, e.g., [1,17]).

The *length* of P is defined to be ℓ, i.e., the number of its variables or equivalently of its instructions. Yet, we remark that a variable may occur several times in the right parts of the instructions P consists of. Such a number of occurrences is called the *fan-out* of that variable. The fan-out of x_ℓ is 0, while the fan-out of any other variable is at least 1 (if no useless instructions occur). The *fan-out* of P is the maximum fan-out of its variables.

Definition 2 The *size* of an SLP is specified by measuring its length and its fan-out.

It is easy to see that a *regular expression* may be seen as an SLP with fan-out 1. In general, due to fan-out power, straight line programs can be exponentially more succinct than regular expressions [17].

We end this section by a quick comment on our way of writing and denoting the type of size increasing when simulating computational models. As the reader noticed, due to their different "hardware", the size of the models under consideration here clearly is not always measured by the same parameters. So, with a slight abuse of terminology we say, e.g., that the size blow-up is exponential in the simulation of the model A with the model B whenever some parameter measuring the size of B is bounded above by some parameter of A appearing at the exponent. An example of this attitude can be seen in the discussion before Proposition 2.

3 Comparing constant length queue automata and finite state automata

We start comparing the descriptional power of constant memory automata with that of classical finite state automata. In [17], it is proved that any constant height NPDA (resp., constant height DPDA) can be converted into an equivalent NFA (resp., DFA) paying by an exponential size increase, this size cost being optimal, that is, necessary in some cases. An analogous result may be obtained for converting constant length NQAs and DQAs:

Theorem 1 *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$*, there exists an equivalent* NFA $A' = \langle Q', \Sigma, \delta', q_0', F'\rangle$ *with* $|Q'| \leq |Q| \cdot |\Gamma^{\leq h}|$*. Moreover, if A is a constant length* DQA *then* A' *is a* DFA.

Proof The key idea is to keep the queue content of A, represented by a string in $\Gamma^{\leq h}$, in the finite control state. The transitions of A' reflect step-by-step the evolution of both state and queue content in A. So, we let our NFA $A' = \langle Q', \Sigma, \delta', q_0', F'\rangle$ have $Q' = Q \times \Gamma^{\leq h}$, $q_0' = [q_0, \vdash]$, $F' = F \times \Gamma^{\leq h}$, and δ' defined as follows, for any $p, q \in Q$, $\omega \in \Gamma^{\leq h}$, $\sigma \in \Sigma \cup \{\varepsilon\}$, and $X \in \Gamma$ (recall that $X^0 = \varepsilon$):

– if $(p, \chi, \omega) \in \delta(q, \sigma, X)$ and $\alpha \in \Gamma^{\leq h-1}$, then $[p, X^e\alpha\omega] \in \delta'([q, X\alpha], \sigma)$, provided that $|X^e\alpha\omega| \leq h$, where $e = 0$ (resp., $e = 1$) if $\chi = D$ (resp., $\chi = K$).

The reader may easily verify that $L(A') = L(A)$, and that this transformation preserves determinism. □

Let us now show the optimality of the exponential simulation costs presented in Theorem 1. Consider the following witness language, for each $h > 0$, each alphabet Γ, and a separator symbol $\sharp \notin \Gamma$:

$$D_{\Gamma,h} = \{w\sharp w : w \in \Gamma^{\leq h}\}.$$

Such a language is accepted by small constant length DQAs, while any accepting NFA must be exponentially larger:

Theorem 2 *For each $h > 0$ and each alphabet Γ:*

(i) The language $D_{\Gamma,h}$ is accepted by a constant length DQA *with* 3 *states, queue alphabet $\Gamma \cup \{\sharp, \vdash\}$, and queue length $h + 1$.*
(ii) Any NFA *accepting the language $D_{\Gamma,h}$ must have at least $|\Gamma^{\leq h}|$ states.*

Proof For point (i), we informally describe the behavior of a constant length DQA A for $D_{\Gamma,h}$, when accepting the input string $w\sharp w \in D_{\Gamma,h}$. First, A stores $w\sharp$ in its queue by remaining in its initial state and using no more than $h+1$ queue cells. Then, by switching to another state, A symbol-by-symbol compares the input suffix w against the queue content by dequeuing any matching symbol. Finally, by reading the sole symbol $\sharp$ in the queue, A reaches a final state by an ε-move. The formal definition of A, correctly managing also inputs not in $D_{\Gamma,h}$, may be easily fixed by the reader.

For point (ii), assume by contradiction an NFA B for $D_{\Gamma,h}$ exists, with less than $|\Gamma^{\leq h}|$ states. Suppose also that any non-final state of B has an outgoing path leading to a final state; otherwise, we can remove the state without altering the accepted language. By counting arguments, there exist $v, w \in \Gamma^{\leq h}$ such that $v \neq w$ and the computation of B on both v and w may end up in the same non-final state q. Now, let α be a word leading B from q to a final state. Consequently, we have that both $v\alpha$ and $w\alpha$ belong to $D_{\Gamma,h}$. From this, we get that $\beta\sharp v\beta = \alpha = \beta\sharp w\beta$ for some $\beta \in \Gamma^{\leq h}$. Hence, $v = w$ against the hypothesis $v \neq w$. □

Now, we investigate the size trade-off between constant length NQAs and DFAs. By Theorem 1, any constant length NQA can be simulated by an equivalent NFA, paying by an exponential size increase. In turn, the classical powerset transformation of NFAs into DFAs (see, e.g., [18,20]) induces another exponential size blow-up. So, we get

Proposition 1 *Any constant length* NQA *with state set Q, queue alphabet Γ, and queue length h can be converted into an equivalent* DFA *with $2^{|Q|\cdot|\Gamma^{\leq h}|}$ states.*

The double-exponential simulation cost pointed out in Proposition 1 is optimal. In fact, for each $h > 0$, each alphabet Γ, and a separator symbol $\sharp \notin \Gamma$, we define the language

$$S_{\Gamma,h} = \{v_1 v_2 \cdots v_r \sharp w_1 w_2 \cdots w_t : v_i, w_j \in \Gamma^h, \text{ for } 1 \leq i \leq r,\ 1 \leq j \leq t, \text{ and } (\cup_{i=1}^{r}\{v_i\}) \cap (\cup_{j=1}^{t}\{w_j\}) \neq \emptyset\}.$$

The following theorem proves that $S_{\Gamma,h}$ can be accepted by a constant length NQA where the number of states and the length of the queue are both linear in h, while any equivalent DFA requires a number of states which cannot be less than double-exponential in h:

Theorem 3 *For each $h > 0$ and each alphabet Γ:*

(i) The language $S_{\Gamma,h}$ is accepted by a constant length NQA *with $O(h)$ states, queue alphabet $\Gamma \cup \{\vdash\}$, and queue length h.*
(ii) Any DFA *accepting the language $S_{\Gamma,h}$ must have at least $2^{|\Gamma^h|}$ states.*

Proof For point (i), informally an NQA A for $S_{\Gamma,h}$ sweeps the first half of the input string while counting modulo h in its finite state control, in order to delimit each block v_i. While doing this, A nondeterministically chooses a block v_i to be stored in the queue, and skips the others until $\sharp$ is reached. From this point on, A sweeps the second half of the input string as done for the first half, but this time it nondeterministically chooses a block w_j to be matched against the block v_i previously stored in the queue. Clearly, the length of the queue is h, and $O(h)$ states suffice to count modulo h and perform queuing/dequeuing operations as described.

For point (ii), assume by contradiction the existence of a DFA A for $S_{\Gamma,h}$ featuring less than $2^{|\Gamma^h|}$ states. By counting arguments, there exist two *different* subsets of Γ^h, say $B = \{x_1, x_2, \ldots, x_m\}$ and $C = \{y_1, y_2, \ldots, y_n\}$, such that A reaches the same state q after processing either the input string $\alpha = x_1 x_2 \cdots x_m$ and the input string $\beta = y_1 y_2 \ldots y_n$. Without loss of generality, assume that $u \in B$ and $u \notin C$. Clearly, the word $\alpha\sharp u$ belongs to $S_{\Gamma,h}$, while the word $\beta\sharp u$ does not. However, starting from q and deterministically processing the same suffix $\sharp u$, we get that A accepts $\alpha\sharp u \in S_{\Gamma,h}$ if and only if it accepts $\beta\sharp u \notin S_{\Gamma,h}$, a contradiction. □

4 The cost of determinizing constant length queue automata

Let us now focus on the cost of removing nondeterminism in constant length NQAs. We are going to prove an optimal exponential cost, in sharp contrast with the realm of constant height NPDAs, where an optimal double-exponential cost is proved in [5].

As a preliminary result, we complete the picture given in the previous section by studying the missing simulation. Precisely, we show that, by having a constant length queue at our disposal, we can remove nondeterminism in finite state automata paying by only a linear size increase.

Theorem 4 *For each* NFA $A = \langle Q, \Sigma, \delta, q_1, F\rangle$*, there exists an equivalent constant length* DQA $A' = \langle Q', \Sigma, \Gamma, \delta', q_0, \vdash, F', h\rangle$ *such that* $\left|Q'\right| \leq 2 \cdot |Q| \cdot |\Sigma| + |\Sigma| + 3$, $|\Gamma| \leq |Q| + 2$, *and* $h \leq 2 \cdot |Q| + 2$.

Proof The key idea is to simulate the behavior of the powerset automaton of A (i.e., the DFA obtained from A by the powerset construction [18,20]) by using the queue of A' to store the set of states in which A might currently be. Each possible transition of A is simulated in A' by consuming the state q at the head of the queue, and enqueuing the set of successors of q on the current input symbol. Basically, A' performs a breadth-first traversal of the computation digraph of A on a given input string, searching for a possible accepting state.

Formally, we define our constant length DQA $A' = \langle Q', \Sigma, \Gamma, \delta', q_0, \vdash, F', h\rangle$ as having:

- $Q' = \{ e_a^q, \tilde{e}_a^q \mid q \in Q, a \in \Sigma \} \cup \{ t_a \mid a \in \Sigma \} \cup \{ q_0, r, r_F \}$,
- $\Gamma = Q \cup \{\sharp, \vdash\}$,
- $F' = \{r_F\}$ if $q_1 \notin F$, otherwise $F' = \{q_0, r_F\}$ if $q_1 \in F$,
- $h = 2 \cdot |Q| + 2$.

Let us now focus on defining the transition function δ'. The first transition initializes the queue, reads the first input symbol, and stores it in the finite state control. So, for every $a \in \Sigma$, we let

$$\delta'(q_0, a, \vdash) = (t_a, D, q_1\sharp\vdash).$$

Roughly speaking, along the computation of A', the queue content will be a string in Γ^* of the form $\alpha\sharp\beta\vdash$, with $\alpha, \beta \in Q^*$, having the following meaning: the prefix α represents the set of states in which A might currently be in, while the factor β represents the set of states A might reach upon reading the current input symbol. This queue content is managed by A' as follows: the first symbol of α (i.e., the state q at the head of the queue) is consumed and its successor states in A are enqueued. To accomplish this task, a queue rotation is required to avoid multiple storing of the same state in the second part β of the queue. To this aim, the transitions on the state e_a^q rotate the queue content until $\sharp$ is reached, while those on $\tilde{e}_a^q$ still rotate the queue content, together with deleting the successors of q already occurring in β. Formally, let $Q = \{q_1, q_2, \ldots, q_n\}$ be the state set of A, and for any $P = \{q_{i_1}, q_{i_2}, \ldots, q_{i_\ell}\} \subseteq Q$ with $i_1 < i_2 < \cdots < i_\ell$ define w_P as the string $q_{i_1} q_{i_2} \cdots q_{i_\ell} \in \Gamma^*$. For any $a \in \Sigma$ and $p, q \in Q$, we let

$$\delta'(t_a, \varepsilon, q) = (e_a^q, D, \varepsilon), \qquad \delta'(e_a^q, \varepsilon, \sharp) = (\tilde{e}_a^q, D, \sharp),$$
$$\delta'(e_a^q, \varepsilon, p) = (e_a^q, D, p), \qquad \delta'(\tilde{e}_a^q, \varepsilon, \vdash) = (t_a, D, w_{\delta(q,a)}\vdash),$$
$$\delta'(\tilde{e}_a^q, \varepsilon, p) = \begin{cases} (\tilde{e}_a^q, D, \varepsilon) & \text{if } p \in \delta(q, a), \\ (\tilde{e}_a^q, D, p) & \text{if } p \notin \delta(q, a). \end{cases}$$

If there is no further state symbol of A in the first part of the queue, i.e., if the head of the queue is $\sharp$, we need to process β upon the next input symbol. To this aim, the queue content is modified from $\sharp\beta\vdash$ to $\beta\sharp\vdash$. We get this by rotating the queue in the states r and r_F, while checking whether some state in F shows up in the queue. If this is the case, A' enters the accepting state r_F at the end of rotation; otherwise, it enters the state r. Formally, for any $a \in \Sigma$ and $q \in Q$, we let

$$\delta'(t_a, \varepsilon, \sharp) = (r, D, \varepsilon),$$
$$\delta'(r, \varepsilon, q) = \begin{cases} (r, D, q) & \text{if } q \notin F, \\ (r_F, D, q) & \text{if } q \in F, \end{cases} \qquad \delta'(r_F, \varepsilon, q) = (r_F, D, q),$$
$$\delta'(r, a, \vdash) = \delta'(r_F, a, \vdash) = (t_a, D, \sharp\vdash).$$

To see that $L(A') = L(A)$, let the DFA A'' be the powerset automaton from A. First, note that ε is accepted by A' if and only if ε is accepted by A''. Now, assume that A'' is in some state $P_1 \subseteq Q$, reads an input symbol $a \in \Sigma$, and goes to successor state $P_2 \subseteq Q$. Further, assume that A', after consuming the input symbol a, is in the state t_a, and its queue content is $w_1\sharp\vdash$, for some permutation w_1 of w_{P_1}—note that this situation is established since the beginning, after the first transition of A'. Then, for each state symbol q in w_1, the DQA A' deletes this symbol from w_1, and adds the state symbols from $\delta(q, a)$ to the second part of the queue between the $\sharp$ and $\vdash$ symbols. When all state symbols from w_1 are processed, the queue content of A' is $\sharp w_2\vdash$, where w_2 is some permutation of w_{P_2}. Now, A' sees the $\sharp$ symbol in the queue and scans the word w_2, trying to reveal some accepting state symbol $q \in F$ by rotating the queue content symbol by symbol. When A' sees the $\vdash$ symbol, the scanning of w_2 is completed. In this situation, A' is in the accepting state r_F if and only if there is an accepting state of A in the state set P_2. Then, the next input symbol is consumed, the new queue content is $w_2\sharp\vdash$, and A' is ready to simulate the next step of A''. Therefore, after completely sweeping the input string, A' is in the accepting state r_F if and only if A'' entered an accepting state.

Clearly, the DQA A' has $2 \cdot |Q| \cdot |\Sigma| + |\Sigma| + 3$ states and $|Q| + 2$ queue symbols. Moreover, notice that at any time the queue contains: any state symbol $q \in Q$ at most once in the first

part and at most once in the second part, at most one $\sharp$ symbol, at most one $\vdash$ symbol. So, the queue length never exceeds $2 \cdot |Q| + 2$. □

As a consequence of Theorem 4, we can settle the claimed optimal exponential size cost for the determinization of constant length NQAs:

Theorem 5 *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$*, there exists an equivalent constant length* DQA $A' = \langle Q', \Sigma, \Gamma', \delta', q_0', \vdash', F', h'\rangle$ *which satisfies* $|Q'| \in O(|Q| \cdot |\Gamma^{\leq h}| \cdot |\Sigma|)$*,* $|\Gamma'| \in O(|Q| \cdot |\Gamma^{\leq h}|)$*, and* $h' \in O(|Q| \cdot |\Gamma^{\leq h}|)$*. Furthermore, the size cost of this conversion is optimal.*

Proof First, we use Theorem 1 to transform the given constant length NQA into an equivalent NFA, paying by an exponential size increase. Then, we use the linear transformation in Theorem 4 to obtain the desired constant length DQA.

To get the optimality of this exponential size blow-up, assume by contradiction a sub-exponential size increase. Then, by Theorem 1, we would get a sub-double-exponential size cost for converting constant length NQAs into DFAs, against the double-exponential optimality pointed out at Theorem 3. □

5 Comparing constant length queue automata and constant height pushdown automata

We begin by showing that constant height pushdown automata can be simulated by constant length queue automata paying by only a linear size increase.

Theorem 6 *For any constant height* NPDA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F, h\rangle$*, there exists an equivalent constant length* NQA A' *with* $2 \cdot |Q|$ *states,* $|\Gamma| + 1$ *queue symbols, and queue length* $h + 1$*. Moreover, if* A *is a constant height* DPDA *then* A' *is a constant length* DQA*.*

Proof The key idea is to maintain the pushdown storage in the queue so that the queue head (resp., tail) represents the symbol at the top (resp., bottom) of the pushdown. The simulation in A' of a move of A works as follows:

(i) the head (corresponding to the top of A) is consumed,
(ii) the string to pile at the top of the pushdown is enqueued, preceded by a special separator symbol $\sharp \notin \Gamma$,
(iii) the whole queue content is rotated symbol by symbol, until $\sharp$ is consumed.

It is easy to see that at the end of this rotation the new queue content reflects the pushdown content in A after the simulated move.

So, we define our constant length NQA $A' = \langle Q', \Sigma, \Gamma \cup \{\sharp\}, \delta', q_0, \bot, F, h+1\rangle$ as having $Q' = Q \cup \{r_q : q \in Q\}$, and δ' stated as follows. Given in A the move $\delta(q, \sigma, Z) \ni (p, \gamma)$, with $\sigma \in \Sigma \cup \{\varepsilon\}$, we let

$$\begin{aligned} \delta'(q, \sigma, Z) &= \{(r_p, D, \sharp\gamma)\}, \\ \delta'(r_p, \varepsilon, X) &= \{(r_p, D, X)\} \quad \text{for any } X \in \Gamma \setminus \{\sharp\}, \\ \delta'(r_p, \varepsilon, \sharp) &= \{(p, D, \varepsilon)\}. \end{aligned}$$

The first transition enqueues the pushed string and prepares the queue rotation which is then accomplished by the second transitions. Finally, the third transition switches the state according to δ.

Clearly, $|Q'| = |Q| + |\{r_q : q \in Q\}| = 2 \cdot |Q|$, while the set of queue symbols has cardinality $|\Gamma| + 1$. Also, notice that the queue rotation process increases by 1 the length of the queue at the first step, due to appending $\sharp$. In conclusion, observe that no nondeterminism is induced by moves from the states in $\{r_q : q \in Q\}$. So, if A is deterministic, then A' is deterministic as well. □

For the reverse conversions, i.e., from constant length queue automata to constant height pushdown automata, we notice that Theorem 1 directly implies an exponential upper bound since a finite automaton can be seen as a pushdown automaton that does not use its own pushdown store. Therefore,

Proposition 2 *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$*, there exists an equivalent constant height* NPDA $A' = \langle Q', \Sigma, \Gamma', \delta', q_0', \bot, F', h'\rangle$ *which satisfies* $|Q'| \leq |Q| \cdot |\Gamma^{\leq h}|$ *and* $\left|\Gamma'\right| = h' = 1$*. Moreover, if A is a constant length* DQA *then A′ is a constant height* DPDA*.*

A corresponding exponential size cost lower bound for converting constant length queue automata to constant height pushdown automata, that is, the optimality of Proposition 2, will be proved later in Proposition 6, Sect. 6.

We end this section, by addressing the size costs for the conversions of constant height NPDAs to constant length DQAs, and constant length NQAs to constant height DPDAs. We show an optimal exponential size cost for the former conversion, and a double-exponential size cost for the latter.

Proposition 3 *For each constant height* NPDA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F, h\rangle$*, there exists an equivalent constant length* DQA $A' = \langle Q', \Sigma, \Gamma', \delta', q_0', \vdash, F', h'\rangle$ *which satisfies* $|Q'| \in O(|Q| \cdot |\Gamma^{\leq h}| \cdot |\Sigma|)$*,* $|\Gamma'| \in O(|Q| \cdot |\Gamma^{\leq h}|)$*, and* $h' \in O(|Q| \cdot |\Gamma^{\leq h}|)$*. Furthermore, the size coast of this conversion is optimal.*

Proof For the size cost upper bound, it suffices to apply Theorem 6 and get from A an equivalent constant length NQA which in turn, according to Theorem 5, can be transformed into an equivalent constant length DQA A' with the claimed size.

For the optimality, we notice that if every constant height NPDA could be transformed to an equivalent constant length DQA with a sub-exponential size blow-up, then we could use the exponential transformation from Proposition 2 to convert the resulting constant length DQA into an equivalent constant height DPDA. Altogether, this would yield a sub-double-exponential size cost for removing nondeterminism in constant height NPDAs, against the results proved in [5]. □

Proposition 4 *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$*, there exists an equivalent constant height* DPDA $A' = \langle Q', \Sigma, \Gamma', \delta', q_0', \bot, F', h'\rangle$ *which satisfies* $|Q'| \leq 2^{|Q| \cdot |\Gamma^{\leq h}|}$ *and* $|\Gamma'| = h' = 1$*. Furthermore, the size cost of this conversion is optimal.*

Proof The size cost upper bound directly follows from Proposition 1. For the optimality, we observe that a sub-double-exponential size cost conversion plus the linear transformation in Theorem 6, from constant height NPDAs to constant length NQAs would yield a sub-double-exponential cost for removing nondeterminism in constant height NPDAs, against the results proved in [5]. □

6 Comparing queue automata and straight line programs

Let us state the cost of turning constant length NQAs into SLPs. By composing Proposition 2 with a result in [17] relating the size of a constant height NPDA with the size of an equivalent SLPs we get:

Proposition 5 *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$*, there exists an equivalent* SLP *of length* $O(|Q|^4 \cdot |\Gamma^{\leq h}|^4 \cdot |\Sigma|)$ *and fan-out* $O(|Q|^2 \cdot |\Gamma^{\leq h}|^2)$.

Proof The exponential size cost is obtained by composing the exponential simulation from constant length NQAs to constant height NPDAs in Proposition 2 with the polynomial transformation given in [17], from constant height NPDAs to SLPs. □

For the optimality of Proposition 5, we consider the following language, for each $h > 0$, each alphabet Γ, and separator symbols $\$, \sharp \notin \Gamma$:

$$L_{\Gamma,h} = \bigcup_{u \in \Gamma^h} \{ (\sharp u)^i \$ \mid i \geq 1 \}.$$

Theorem 7 *For each* $h > 0$ *and each alphabet* Γ*:*

(i) The language $L_{\Gamma,h}$ *is accepted by a constant length* DQA *with* $O(h)$ *states, queue alphabet* $\Gamma' = \Gamma \cup \{\vdash, \sharp\}$*, and queue length* $h + 1$.
(ii) Any SLP *accepting the language* $L_{\Gamma,h}$ *must have at least* $|\Gamma^h|$ *variables.*

Proof A constant length DQA for $L_{\Gamma,h}$ stores the prefix $\sharp u$ of a given input word in its queue, while checking that $|u| = h$. This requires $O(h)$ states. Then, it matches its queue content against the rest of the input word by rotating the queue symbol by symbol, rejecting whenever a bad input format problem or a wrong symbol match is revealed. Upon reading the input symbol $\$$ and the symbol $\sharp$ at the head of the queue, the DQA enters a final state.

Let us now switch to SLPs for $L_{\Gamma,h}$, and first introduce some terminology. In an SLP, we call *star-variable* any variable x occurring on the left-hand side in a *star-instruction* of the form $x := y^*$. Moreover, we denote by $L(x)$ the language represented by the regular expression computed in x.

Thus, let P be an SLP computing a regular expression for $L_{\Gamma,h}$, and let P' be the SLP obtained from P by replacing every star-instruction $x := y^*$ by the instruction $x := \varepsilon$. Clearly, P' describes a finite language for which we let m be the length of the longest word. It is easy to see that, for any word $z \in L_{\Gamma,h}$ with $|z| > m$, the SLP P must make use of a star-variable producing some non-empty factor of z. By applying this observation to the word $z_u = (\sharp u)^m \$ \in L_{\Gamma,h}$, with $u \in \Gamma^h$, we get the existence of a star-variable x_u in P such that:

(i) $z_u = z_{u,1} z_{u,2} z_{u,3}$ with $\varepsilon \neq z_{u,2} \in L(x_u)$, and
(ii) for all $z'_{u,2} \in L(x_u)$, we have $z_{u,1} z'_{u,2} z_{u,3} \in L_{\Gamma,h}$.

The non-empty factor $z_{u,2}$ must contain at least one $\sharp$ symbol. Otherwise, P would describe a word with a factor $\omega \in \Gamma^*$ satisfying $|\omega| > h$, which cannot belong to $L_{\Gamma,h}$. If $z_{u,2}$ contains at least two $\sharp$ symbols, then it contains the factor $\sharp u \sharp$. If $z_{u,2}$ contains only one $\sharp$ symbol, i.e., if $z_{u,2} = v_1 \sharp v_2$ for $v_1, v_2 \in \Gamma^*$, then v_1 (resp., v_2) is a suffix (resp., prefix) of u. In fact, since $z_{u,2}^2 \in L(x_u)$, it must be $v_2 v_1 = u$.

Thus, we have shown that, for every word $u \in \Gamma^h$, there exists a star-variable x_u such that $L(x_u)$ contains a word having $\sharp u \sharp$ as a factor. If P has less than $|\Gamma^h|$ variables, then

clearly there exist $u, v \in \Gamma^h$, with $u \neq v$, satisfying $x_u = x_v$. This implies that the language $L(x_u)$ contains words that have either $\sharp u \sharp$ and $\sharp v \sharp$ as factors. Moreover, since $L(x_u)$ is closed under star operation, we get that P describes words of the form $\alpha\sharp u\sharp\beta\sharp v\sharp\gamma \notin L_{\Gamma,h}$, for $\alpha, \beta, \gamma \in (\Gamma \cup \{\sharp, \$\})^*$, a contradiction. Thus, P must have at least $|\Gamma^h|$ variables. □

As a consequence of Theorem 7, we get the optimality of the exponential size cost conversions of constant length queue automata to constant height pushdown automata addressed in Proposition 2:

Proposition 6 *The exponential conversions from constant length* DQAs *to constant height* NPDAs*, and from constant length* NQAs *(resp.,* DQAs*) to constant height* NPDAs *(resp.,* DPDAs*) in Proposition 2 are optimal.*

Proof Assume, by contradiction, a sub-exponential size cost conversion from constant length DQAs to constant height NPDAs. Then, by the polynomial size cost conversion of constant height NPDAs to SLPs shown in [17], one could also convert any constant length DQA to an equivalent SLP of sub-exponential size, against Theorem 7. We can argue similarly for the other conversions. □

We conclude by converting SLPs to constant length NQAs or DQAs:

Proposition 7 *The size blow-up for converting* SLPs *to equivalent constant length* DQAs *(resp.,* NQAs*) is exponential (resp., linear). Furthermore, the former exponential transformation is optimal.*

Proof In [17], a linear size cost transformation from SLPs to constant height NPDAs is presented. Together with the linear size cost conversion from constant height NPDAs to constant length NQAs established in Theorem 6, this yields a linear size cost conversion from SLPs to constant length NQAs. In turn, Theorem 5 implies an exponential size cost upper bound for converting of SLPs to constant length DQAs.

For the optimality of this latter exponential size blow-up, assume by contradiction a sub-exponential size cost conversion exists, from SLPs to constant length DQAs. Then, by a preliminary size cost polynomial transformation from constant height NPDAs to SLPs, plus a successive exponential size cost conversion from constant length DQAs to constant height DPDAs, one could get a sub-double-exponential size cost for removing nondeterminism in constant height NPDAs, against results proved in [5]. □

7 Summary of simulation costs

For reader's ease of mind, we sum up in Fig. 1 the main relations among the sizes of the different types of formalisms for regular languages considered in this paper. Such size relations are briefly commented below the figure.

Linear and polynomial size blow-ups. The linear size costs of the transformations DFA $\to$ NFA, DFA $\to$ h-DQA, NFA $\to$ h-NQA, DFA $\to$ h-NQA, h-DQA $\to$ h-NQA, and h-DPDA $\to$ h-NPDA are trivial. Let us focus on the other linear and polynomial costs:

- NFA $\to$ h-DQA: the linear cost comes from Theorem 4.
- SLP $\to$ h-NQA: the linear cost comes from Proposition 7.
- h-DPDA $\to$ h-DQA, h-DPDA $\to$ h-NQA, and h-NPDA $\to$ h-NQA: the linear costs come from Theorem 6.

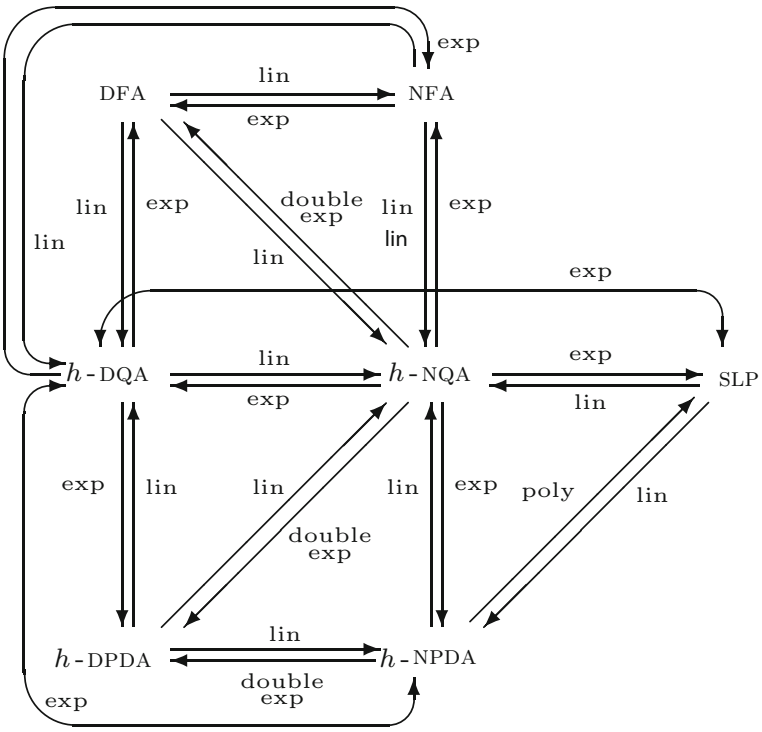

Fig. 1 The size costs of simulations among different types of formalisms defining regular languages. Here h-DPDA (h-NPDA, resp.) denotes a constant height DPDA (NPDA, resp.), while h-DQA (h-NQA, resp.) denotes a constant length DQA (NQA, resp.). An arc labeled by lin (poly, exp, double exp, resp.) from a vertex A to a vertex B means that, given a representation of type A, we can build an equivalent representation of type B, paying by a linear (polynomial, exponential, double-exponential, resp.) increase in the size

– SLP $\to$ h-NPDA: the linear cost comes from [17].
– h-NPDA $\to$ SLP: the polynomial cost comes from [17].

Exponential and double-exponential size blow-ups.

– NFA $\to$ DFA: the exponential cost is known from [31], its optimality from [30].
– h-DQA $\to$ DFA, h-DQA $\to$ NFA, and h-NQA $\to$ NFA: the exponential costs come from Theorem 1, their optimality from Theorem 2.
– h-NQA $\to$ DFA: the double-exponential cost comes from Proposition 1, its optimality from Theorem 3.
– h-DQA $\to$ SLP: the exponential cost comes from Proposition 5, its optimality from Theorem 7.
– SLP $\to$ h-DQA: the optimal exponential cost comes from Proposition 7.
– h-NQA $\to$ h-DQA: the optimal exponential cost comes from Theorem 5.
– h-NQA $\to$ SLP: the exponential cost comes from Proposition 5, its optimality from Theorem 7.
– h-DQA $\to$ h-DPDA, h-DQA $\to$ h-NPDA, and h-NQA $\to$ h-NPDA: the exponential costs come from Proposition 2, their optimality from Proposition 6.
– h-NPDA $\to$ h-DQA: the optimal exponential cost comes from Proposition 3.
– h-NQA $\to$ h-DPDA: the optimal double-exponential cost comes from Proposition 4.

– h-NPDA $\rightarrow$ h-DPDA: the optimal double-exponential cost comes from [5].

8 Boolean language operations on constant length queue automata

Let us now study the size cost of implementing boolean language operations on constant length NQAs and DQAs. This is a commonly investigated topic in descriptional complexity, for formalisms defining regular languages. For instance, for constant height NPDAs and DPDAs, results can be found in [3,4,6]. Indeed, also for nonclassical language acceptors, such as different variants of quantum finite automata, several results on the size complexity of boolean language operations are established (see, e.g., [7–10,12,13]).

To simplify our constructions, sometimes it will be useful to assume that our queue automata enqueue at most one symbol at any move, that is, any transition $(p, \chi, \omega) \in \delta(q, \sigma, X)$ satisfies $|\omega| \leq 1$. In this case, the queue length changes at most by 1 at any move, and we say that the queue automaton is in *normal form*.

The next technical lemma shows that any constant length queue automaton can be converted in normal form, by preserving determinism. To suitably evaluate the size cost of this conversion, we adopt a size measure accounting for the "complexity" of the instructions of the queue automaton.

According to [18], the size of an automaton M can be defined as the length of a string describing its transition function. For a queue automaton, if the ith transition of M is $\delta(q, \sigma, X) \ni (p, \chi, Y_1 \dots Y_k)$, with $k \geq 0$ and $Y_s \in \Gamma$, it can be written down as a string $t_i = q\, \sigma\, X p\, \chi\, Y_1 \dots Y_k$. (Here we assume, without loss of generality, that Q, Σ, Γ and $\{D, K\}$ are mutually disjoint sets. This makes decoding of transitions unambiguous.) So, globally M can be written down as $t_1 \cdots t_m \in (Q \cup \Sigma \cup \Gamma \cup \{D, K\})^*$, a string listing all machine's instructions one after another. By charging "1" for the constant part in each transition, we define the size of M as

$$|M| = \sum_{(q,\sigma,X) \in Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma} \; \sum_{(p,\chi,\omega) \in \delta(q,\sigma,X)} \max\{|\omega|, 1\}\,.$$

As the reader may easily verify, we have that $|M| \in O(|Q|^2 \cdot |\Gamma|^2)$.

We are now ready to establish the size cost of converting constant length queue automata into normal form:

Lemma 1 *Given a constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$, *there exists an equivalent constant length* NQA $A' = \langle Q', \Sigma, \Gamma, \delta', q_0, \vdash, F, h\rangle$ *in normal form, with* $|Q'| \leq |A| + 1$. *Moreover, if A is deterministic, then* A' *is deterministic as well.*

Proof Consider an instruction $\rho = \delta(q, \sigma, X) \ni (p, \chi, \gamma)$ of A, with $\gamma \in \Gamma^*$. If $|\gamma| \leq 1$, then we let $\delta'(q, \sigma, X) \ni (p, \chi, \gamma)$. Otherwise, assuming $\gamma = Y_1 \dots Y_k$ with $k > 1$, we add $k - 1$ new states $s_1, \dots, s_{k-1}$ and, for $1 \leq i \leq k - 2$ and all $Z \in \Gamma$, we translate ρ into the following instructions:

$$\begin{aligned} \delta'(q, \sigma, X) &\ni (s_1, \chi, Y_1), \\ \delta'(s_i, \varepsilon, Z) &= \{(s_{i+1}, K, Y_{i+1})\}, \\ \delta'(s_{k-1}, \varepsilon, Z) &= \{(p, K, Y_k)\}. \end{aligned}$$

Let us count the number of states required by A'. For the instruction ρ, the automaton A' uses the states $q, s_1, \dots, s_{|\gamma|-1}, p$. If q is reachable by some nonempty input string, then it is already counted in some other instruction translation. Therefore, simulating the instruction

ρ requires at most $\max\{|\gamma|, 1\}$ additional states. The only state that might not be counted is the initial state q_0. Altogether, we get $|Q'| \leq |A| + 1$. □

Another simplification we need for combining two constant length NQAs in order to implement boolean language binary operations, is to assume that such automata have the same queue length. The evaluation of the size blow-up to implement this assumption is the subject of the following:

Lemma 2 *Given a constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$ *in normal form, for any* $h' > h$ *we can construct an equivalent constant length* NQA A' *in normal form, having queue length* h'*, queue alphabet cardinality bounded by* $|\Gamma| + 1$*, and featuring* $\min\{h \cdot |Q|,\ |Q| \cdot (|\Gamma| + 1) + h' - h\}$ *states. Moreover, if* A *is deterministic, then* A' *is deterministic as well.*

Proof First, we notice that a trivial step-by-step simulation of A by A' using a queue of length h' would not be able to reproduce the situation in which A rejects by attempting to enqueue more than h symbols. Thus, one possibility is to build A' by integrating an integer counter within the finite state control of A, keeping track of the current queue length and immediately rejecting whenever the counter attempts to exceed h. This clearly implies the usage of $h \cdot |Q|$ states in A'.

Another possibility is that A' initially inserts $h' - h$ dummy symbols in the queue, which will never be deleted. This clearly reduces A''s queue capacity to h, and requires $h' - h$ new states to be implemented. Along the computation, A' maintains the simulated queue content of A before the dummy symbols by using a queue rotation subroutine. For this subroutine, we add new states of the form r_p^Y storing the state p to be entered and the symbol Y to be moved from the head to the tail of the queue (see, e.g., proofs of Theorems 4 and 6). This implies the usage of $|Q| + |Q| \cdot |\Gamma| + h' - h$ states in A' and a queue alphabet which is Γ equipped with a dummy symbol. □

We are now ready to implement boolean language binary operations on constant length NQAs which, by considering Lemmas 1 and 2 above, are assumed to be in normal form and working with the same queue length. We begin by considering the intersection:

Theorem 8 *Given two constant length* NQAs $A_1 = \langle Q_1, \Sigma, \Gamma_1, \delta_1, c_1, \vdash, F_1, h\rangle$ *and* $A_2 = \langle Q_2, \Sigma, \Gamma_2, \delta_2, c_2, \vdash, F_2, h\rangle$ *in normal form, we can construct a constant length* NQA A *accepting* $L(A_1) \cap L(A_2)$*, having queue length* $h + 1$*, queue alphabet cardinality* $(|\Gamma_1| + 1) \cdot (|\Gamma_2| + 1) + 1$*, and featuring* $|Q_1| \cdot (|\Gamma_1| + 1)^2 \cdot |Q_2| \cdot (|\Gamma_2| + 1)^2$ *states.*

Proof The key idea is that A simulates in parallel the behavior of A_1 and A_2, using its queue divided into two tracks. To this aim, the queue symbols of A consist of pairs of symbols from $\Gamma_1 \times \Gamma_2$. Analogously, the states of A come from $Q_1 \times Q_2$. The two components of the elements of these two sets are responsible of simulating the computation steps of A_1 and A_2 on the corresponding tracks. Since the queue operations of A_1 and A_2 are generally not synchronized, a special *blank* queue symbol $b \notin \Gamma_1 \cup \Gamma_2$ is inserted at the end of the shortest track to allow tracks aligning. Every enqueuing operation requires a rotation subroutine in order to keep the b's at the end of the shortest track. To this purpose, a special queue symbol $(\sharp, \sharp)$, with $\sharp \notin \Gamma_1 \cup \Gamma_2$, is introduced, together with new elements that will be used as state components of A during the rotation subroutine. Namely, for $i \in \{1, 2\}$, we let:

- $R_i = \{r_p^Y : p \in Q_i, Y \in \Gamma_i\}$. An element r_p^Y stores the state p to be entered and the queue symbol Y to be enqueued at the end of the non-blank part of the corresponding track.[2]
- $S_i = \{s_p^X : p \in Q_i, X \in \Gamma_i\}$. An element s_p^X stores the state p to be entered and the queue symbol X to be enqueued at the next rotation step during the rotation subroutine. This is required whenever the operations of A_1 and A_2 on the queue are not synchronized.
- $T_i = \{t_p^{Y,X} : p \in Q_i, X, Y \in \Gamma_i\}$. An element $t_p^{Y,X}$ stores the state p to be entered, the queue symbol Y to be enqueued at the end of the non-blank part of the corresponding track, and the queue symbol X to be enqueued at the next rotation step during the subroutine.

Formally, we define $A = \langle Q, \Sigma, \Gamma, \delta, (c_1, c_2), (\vdash, \vdash), F, h+1\rangle$ with

- $Q = \{(p_1, p_2) : p_i \in Q_i \cup R_i \cup S_i \cup T_i,\ i \in \{1, 2\}\}$,
- $\Gamma = (\Gamma_1 \cup \{b\}) \times (\Gamma_2 \cup \{b\}) \cup \{(\sharp, \sharp)\}$, for $b, \sharp \notin \Gamma_1 \cup \Gamma_2$,
- $F = F_1 \times F_2$.

The transition function δ simulates δ_1 and δ_2 on the two tracks, and performs the rotation subroutine whenever required.

Particular attention must be paid to handle ε-moves. If one of the two automata A_1 or A_2 requires to perform an ε-move, the other automaton must be synchronized by replicating an ε-move too, in which state and queue are not modified. To this aim, we add the following rules to δ_i, for $i \in \{1, 2\}$, which can be simulated by δ. For any $q \in Q_i$ and $X \in \Gamma_i$, we let:

$$\delta_i(q, \varepsilon, X) \ni (q, K, \varepsilon). \tag{1}$$

For the sake of brevity, here we formalize the rules in δ for the case DELETE- - KEEP, i.e., simulating the transitions

$$\delta_1(q_1, \sigma, X_1) \ni (p_1, D, Y_1) \text{ and } \delta_2(q_2, \sigma, X_2) \ni (p_2, K, Y_2),$$

where $\sigma \in \Sigma \cup \{\varepsilon\}$ and $Y_1, Y_2 \neq \varepsilon$. We define δ, for any $(Z_1, Z_2) \in \Gamma_1 \times \Gamma_2$, as:

$$\begin{aligned}
\delta((q_1, q_2), \sigma, (X_1, X_2)) &\ni ((r_{p_1}^{Y_1}, t_{p_2}^{Y_2,X_2}), D, (\sharp, \sharp)),\\
\delta((r_{p_1}^{Y_1}, t_{p_2}^{Y_2,X_2}), \varepsilon, (Z_1, Z_2)) &\ni ((r_{p_1}^{Y_1}, t_{p_2}^{Y_2,Z_2}), D, (Z_1, X_2)),\\
\delta((r_{p_1}^{Y_1}, t_{p_2}^{Y_2,Z}), \varepsilon, (\sharp, \sharp)) &\ni ((p_1, p_2), D, (Y_1, Z)(Y_2, b)),
\end{aligned}$$

where the last rule takes place whenever the two tracks have the same length. Instead, in case of different queue lengths, we have two different sets of rules depending on whether a b symbol appears in the track dedicated to A_1 or in that for A_2. So, for any $X \in \Gamma_2$, we let:

$$\begin{aligned}
\delta((r_{p_1}^{Y_1}, t_{p_2}^{Y_2,Z}), \varepsilon, (b, X)) &\ni ((p_1, r_{p_2}^{Y_2}), D, (Y_1, Z)(b, X)),\\
\delta((p_1, r_{p_2}^{Y_2}), \varepsilon, (b, X)) &\ni ((p_1, r_{p_2}^{Y_2}), D, (b, X)),\\
\delta((p_1, r_{p_2}^{Y_2}), \varepsilon, (\sharp, \sharp)) &\ni ((p_1, p_2), D, (b, Y_2)),
\end{aligned}$$

while, for any $X \in \Gamma_1$, we let:

$$\begin{aligned}
\delta((r_{p_1}^{Y_1}, t_{p_2}^{Y_2,Z}), \varepsilon, (X, b)) &\ni ((r_{p_1}^{Y_1}, r_{p_2}^{Y_2}), D, (X, Z)),\\
\delta((r_{p_1}^{Y_1}, r_{p_2}^{Y_2}), \varepsilon, (X, b)) &\ni ((r_{p_1}^{Y_1}, p_2), D, (X, Y_2)),\\
\delta((r_{p_1}^{Y_1}, p_2), \varepsilon, (X, b)) &\ni ((r_{p_1}^{Y_1}, p_2), D, (X, b)),
\end{aligned}$$

[2] Rotating one track symbol from head to tail cannot be done in a single move since two not-necessarily synchronized tracks must be managed. This is the reason why R_i's sets are provided.

$$\delta((r_{p_1}^{Y_1}, p_2), \varepsilon, (\sharp, \sharp)) \ni ((p_1, p_2), D, (Y_1, b)).$$

The reader may fill the definition of δ for the cases $Y_1 = \varepsilon$ or/and $Y_2 = \varepsilon$. We simply notice that for $Y_2 = \varepsilon$, instead of having the second component of the states in the form $t_{p_2}^{Y_2,Z}$, we have $s_{p_2}^{Z}$ since we do not record Y_2 to be inserted.

The rules in δ for simulating moves of type KEEP- - DELETE are managed symmetrically, while moves KEEP- - KEEP and DELETE- - DELETE are replicated simply by parallel simulations (in these two latter cases, involved states are only from the set $(Q_1 \cup R_1) \times (Q_2 \cup R_2)$).

It is easy to see that A is a constant length NQA, with queue length at most $h+1$. Moreover, since each component of the states of A is from the set $Q_i \cup R_i \cup S_i \cup T_i$, we have the claimed number of states. □

We observe that the two constant length NQAs considered in Theorem 8 for intersection enjoy particular properties, e.g., being in normal form and operating with the same queue length. However, by considering the transformation size costs in Lemma 1 and Lemma 2, the reader may easily verify that the state cost of implementing the intersection for two *general* constant length NQAs turns out to be at most the square of the state cost pointed out in Theorem 8, multiplied by the length of the shortest queue among the two involved constant length NQAs.

Let us now focus on intersecting constant length DQAs. It is not hard to see that the construction provided in Theorem 8 does not preserve determinism. However, such a construction can be adapted to implement the intersection of two *realtime* constant length DQAs by a constant length DQA. We recall that a queue automaton is said to be realtime whenever it does not present ε-moves (see, e.g., [14]):

Corollary 1 *Given two realtime constant length* DQAs $A_1 = \langle Q_1, \Sigma, \Gamma_1, \delta_1, c_1, \vdash, F_1, h\rangle$ *and* $A_2 = \langle Q_2, \Sigma, \Gamma_2, \delta_2, c_2, \vdash, F_2, h\rangle$ *in normal form, we can construct a constant length* DQA A *accepting* $L(A_1) \cap L(A_2)$*, having queue length* $h+1$*, queue alphabet cardinality* $(|\Gamma_1|+1)\cdot(|\Gamma_2|+1)+1$*, and featuring* $|Q_1|\cdot(|\Gamma_1|+1)^2\cdot|Q_2|\cdot(|\Gamma_2|+1)^2$ *states.*

Proof We can consider the construction provided in the proof of Theorem 8, by simply observing that there the only rules adding nondeterminism are the ones displayed in Eq. (1) for handling ε-moves synchronization. Since A_1 and A_2 are realtime, we here do not need these additional rules any more. □

For general realtime constant length DQAs, again by considering Lemma 1 and Lemma 2, the reader may easily verify that the state cost of implementing intersection turns out to be at most the square of the state cost pointed out in Corollary 1, multiplied by the length of the shortest queue among the two involved constant length DQAs.

Instead, in case of two nonrealtime constant length DQAs, the idea is to turn one of the two devices into an equivalent DFA according to Theorem 1, remove ε-moves from the DFA, and finally embed the DFA into the finite control of the other DQA. The resulting constant length DQA implements the intersection with a number of states bounded by $|Q_1|\cdot|Q_2|\cdot|\Gamma_2^{\leq h_2}|$, where Q_1, Q_2, Γ_2, h_2 have the usual meaning.

Now, we turn to implementing the union. For this operation, the construction is simpler since we can exploit nondeterminism:

Theorem 9 *Given two constant length* NQAs $A_1 = \langle Q_1, \Sigma, \Gamma_1, \delta_1, c_1, \vdash, F_1, h\rangle$ *and* $A_2 = \langle Q_2, \Sigma, \Gamma_2, \delta_2, c_2, \vdash, F_2, h\rangle$*, we can construct a constant length* NQA A *recognizing* $L(A_1)\cup L(A_2)$*, with queue length* h*, queue alphabet* $\Gamma_1 \cup \Gamma_2$*, and* $|Q_1|+|Q_2|+1$ *states.*

Proof Without loss of generality, we can assume that Q_1 and Q_2 are disjoint sets. The states of A are from the set $Q_1 \cup Q_2$, plus a new initial state c, the queue alphabet is $\Gamma_1 \cup \Gamma_2$, and the set of final states is $F_1 \cup F_2$. As a first step, A performs a nondeterministic ε-move from c leading to enter either c_1 or c_2. Then, A simulates δ_1 on states from Q_1, and δ_2 on states from Q_2. It is not hard to see that A accepts whenever either A_1 or A_2 accepts. □

In case of two constant length NQAs with queues of different lengths, by Lemma 2 one may easily obtain that the state cost for their union turns out to be at most $|Q_1|+h_2\cdot|Q_2|+1$, where Q_1, Q_2, h_2 have the usual meaning.

For implementing the union in the realtime deterministic case, we can use the same construction for intersection, addressed in the proof Corollary 1, with small modifications. Such modifications are basically due to handling situations in which one of the two queue automata rejects before consuming the whole input. To this aim, we present a technical lemma:

Lemma 3 *Given a realtime constant length* DQA $A = \langle Q, \Sigma, \Gamma, \delta, c, \vdash, F, h\rangle$*, we can construct an equivalent realtime constant length* DQA $A' = \langle Q', \Sigma, \Gamma, \delta', c, \vdash, F', h\rangle$ *with* $|Q'| \leq h\cdot|Q|+2$*, which always consumes the whole input before halting. Moreover, if A is in normal form, then* A' *is in normal form as well.*

Proof Being A realtime, there cannot exist infinite loops due to ε-moves. The only situations where A might halt before the end of the input string are the following:

(i) queue underflow and overflow,
(ii) undefined δ for some configurations.

To fix situation (i), we store the current length of the queue in the finite state control of A', i.e., the states of A' has the form $(q, \ell) \in Q \times \{1, \ldots, h\}$. Underflow and overflow situations can be detected from the values of ℓ, and can be managed by redirecting the corresponding computation to a new trap state rej. A particular situation might arise, where the queue length of A is zero, the input has been completely scanned, and A enters an accepting state. To handle this situation, we equip A' with a new accepting state acc and, for $\delta(q, a, Z) = (p, D, \varepsilon)$ with $p \in F$, we let

$$\delta'((q, 1), a, Z) = (acc, K, \varepsilon) \text{ and } \delta'(acc, \sigma, X) = (rej, K, \varepsilon),$$

for any $\sigma \in \Sigma$ and $X \in \Gamma$. When entering the state rej, we consume the whole input by the rules $\delta'(rej, \sigma, X) = (rej, K, \varepsilon)$.

To fix situation (ii), we complete the transition function by adding new rules $\delta'(q, \sigma, X) = (rej, K, \varepsilon)$, whenever $\delta(q, \sigma, X)$ is not defined.

Therefore, $Q' = Q \times \{1, \ldots, h\} \cup \{rej, acc\}$, $F' = F \times \{1, \ldots, h\} \cup \{acc\}$, and δ' is a total function that simulates δ on states in Q' by updating the counter and preventing queue underflow and overflow as above addressed. □

We are now ready to show the size cost of the union operation in the realtime deterministic case:

Corollary 2 *Given two realtime constant length* DQAs $A_1 = \langle Q_1, \Sigma, \Gamma_1, \delta_1, c_1, \vdash, F_1, h\rangle$ *and* $A_2 = \langle Q_2, \Sigma, \Gamma_2, \delta_2, c_2, \vdash, F_2, h\rangle$ *in normal form, we can construct a constant length* DQA A *accepting* $L(A_1) \cup L(A_2)$*, having queue length* $h + 1$*, queue alphabet cardinality* $(|\Gamma_1|+1)\cdot(|\Gamma_2|+1)+1$*, and* $(h\cdot|Q_1|+2)\cdot(|\Gamma_1|+1)^2\cdot(h\cdot|Q_2|+2)\cdot(|\Gamma_2|+1)^2$ *states.*

Proof The construction of A works out similarly to that in the proof of Corollary 1. However, we here have to handle cases in which one of the two simulated automata A_1 or A_2 halts before consuming the whole input and the other accepts. To this aim, we first transform A_1 and A_2 into A'_1 and A'_2 according to Lemma 3. Then, to get A, we compose A'_1 and A'_2 as in Corollary 1, except for the set of final states which is now defined to be $(F'_1 \times Q'_2) \cup (Q'_1 \times F'_2)$. □

For general realtime constant length DQAs, the reader may easily verify that the state cost of implementing the union turns out to be the same as the one above discussed for intersecting general realtime constant length DQAs, multiplied by the factor $h_1 \cdot h_2$. This latter factor follows from the construction in Lemma 3, aimed to accept or reject at the end of the input only.

Instead, in case of non-realtime devices, we use the same construction above addressed for implementing the intersection of non-realtime constant length DQAs, thus yielding the same state complexity.

Finally, we consider the size cost of implementing the complement operation:

Theorem 10 *Given a constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h \rangle$*, we can construct a constant length* DQA A' *accepting* $L(A)^c$*, with queue length* $2 \cdot (|Q| \cdot |\Gamma^{\leq h}|) + 4$*, queue alphabet cardinality* $|Q| \cdot |\Gamma^{\leq h}| + 3$*, and* $2 \cdot (|Q| \cdot |\Gamma^{\leq h}| + 1) \cdot |\Sigma| + |\Sigma| + 3$ *states.*

Proof First, we transform A into an equivalent NFA, according to Theorem 1. The transition function of the obtained NFA might be partial and containing ε-moves. Therefore, we complete this function by adding transitions to a new trap state in which the remaining part of the input is consumed, and remove ε-moves by standard tools (see, e.g., [18,20]) without increasing the number of states.

Then, we use Theorem 4 to convert the obtained NFA to an equivalent constant length DQA. Considering the above-mentioned properties of the NFA to be converted and how the construction of Theorem 4 works, it is not hard to see that the resulting constant length DQA always sweeps the whole input, accepting or rejecting at the end only. So, by swapping accepting and rejecting states, we obtain a constant length DQA for $L(A)^c$, with the claimed features. □

For complementing a realtime constant length DQA, it is enough to use Lemma 3 to get an equivalent automaton consuming the whole input, and then swap accepting and rejecting states of this automaton. Thus, we have

Corollary 3 *Given a realtime constant length* DQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h \rangle$*, we can construct a realtime constant length* DQA *for* $L(A)^c$*, having queue length* h*, queue alphabet* Γ*, and featuring* $h \cdot |Q| + 2$ *states.*

In case of non-realtime constant length DQAs, the state cost for the complement remains the same. In fact, before applying the construction used for proving Corollary 3, it suffices to precompute and remove infinite loops due to ε-moves.

9 Conclusions

In this paper, we have considered the notion of a *constant length queue automaton*—i.e., a traditional queue automaton with a built-in constant limit on the length of its queue—as a formalism for representing regular languages.

We have shown that the descriptional power of constant length queue automata is higher than that of several formalisms for defining regular languages by providing optimal exponential and double-exponential size gaps.

In particular, we have proved that constant height pushdown automata can be simulated by constant length queue automata paying only by a linear size increase, and that removing nondeterminism in constant length queue automata requires an optimal exponential size blow-up, against the optimal double-exponential cost for determinizing constant height pushdown automata. This proves the descriptional advantages of using a queue instead of a pushdown in the realm of constant memory machines.

Finally, we have investigated the size cost of implementing boolean language operations on deterministic and nondeterministic constant length queue automata.

Among possible future lines of research, one may investigate restricted variants of constant memory automata, for instance devices working on *unary*, i.e., single-letter, input alphabets (see, e.g., [11,15,26,27]), or input-driven devices [25]. We also would like to emphasize the interest in *two-way* devices [28].

Concerning the investigation of the size cost of language operations on constant length queue automata, it would be interesting to consider further typical operations on regular languages, such as Kleene's star, reversal, quotient, shuffle, homomorphisms and inverse homomorphisms, *etc.* Moreover, one could extend the investigation on language operations for restricted models, such as realtime constant length NQAs. Also the size optimality should be proved or improved constructions should be designed, for the language operations considered in this paper.

Acknowledgements The authors wish to thank the anonymous referees for valuable comments and remarks.

Funding Open access funding provided by Università degli Studi di Milano within the CRUI-CARE Agreement.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Allevi, E., Cherubini, A., Crespi Reghizzi, S.: Breadth-first phrase-structure grammars and queue automata. In: Proceedings of the Mathematics Foundations of Computer Science, Lecture Notes in Computer Science, vol. 324, pp. 162–170. Springer (1988)
3. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: The size-cost of Boolean operations on constant height deterministic pushdown automata. Theor. Comput. Sci. **449**, 23–36 (2012)
4. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Boolean language operations on nondeterministic automata with a pushdown of constant height. In: Proceedings of the Computer Science Symposium Russia, Lecture Notes in Computer Science, vol. 7913, pp. 100–111. Springer (2013)
5. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Removing nondeterminism in constant height pushdown automata. Inf. Comput. **237**, 257–267 (2014)
6. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Boolean language operations on nondeterministic automata with a pushdown of constant height. J. Comput. Syst. Sci. **90**, 99–114 (2017)

7. Bertoni, A., Mereghetti, C., Palano, B.: Golomb rulers and difference sets for succinct quantum automata. Int. J. Found. Comput. Sci. **14**, 871–888 (2003)
8. Bertoni, A., Mereghetti, C., Palano, B.: Trace monoids with idempotent generators and measure-only quantum automata. Nat. Comput. **9**, 383–395 (2010)
9. Bianchi, M.P., Mereghetti, C., Palano, B.: Complexity of Promise Problems on Classical and Quantum Automata, Computing with New Resources—Essays Dedicated to J. Gruska on the Occasion of His 80th Birthday, pp. 161–175. Springer (2014)
10. Bianchi, M.P., Mereghetti, C., Palano, B.: On the power of one-way finite automata with quantum and classical states. In: Proceedings of the Implementation of Automation Applications, Lecture Notes in Computer Science, vol. 8587, pp. 84–97. Springer (2014)
11. Bianchi, M.P., Mereghetti, C., Palano, B., Pighizzini, G.: On the size of unary probabilistic and nondeterministic automata. Fund. Inf. **112**, 119–135 (2011)
12. Bianchi, M.P., Mereghetti, C., Palano, B.: Quantum finite automata: advances on Bertoni's ideas. Theor. Comput. Sci. **664**, 39–53 (2017)
13. Bianchi, M.P., Palano, B.: Behaviours of unary quantum automata. Fund. Inf. **104**, 1–15 (2010)
14. Cherubini, A., Citrini, C., Crespi Reghizzi, S., Mandrioli, D.: QRT FIFO automata, breadth-first grammars and their relations. Theor. Comput. Sci. **85**, 171–203 (1991)
15. Chrobak, M.: Finite automata and unary languages. Theor. Comput. Sci. **47**, 149–158, (1986); Corrigendum, ibid **302**, 497–498 (2003)
16. Gao, Y., Moreira, N., Reis, R., Yu, S.: A survey on operational state complexity. J. Autom. Lang. Comb. **21**, 251–310 (2017)
17. Geffert, V., Mereghetti, C., Palano, B.: More concise representation of regular languages by automata and regular expressions. Inf. Comput. **208**, 385–94 (2010)
18. Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Reading (1978)
19. Holzer, M., Kutrib, M.: Descriptional complexity—an introductory survey. In: Scientific Applications of Language Methods, pp. 1–58. Imperial College Press (2010)
20. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (2001)
21. Jakobi, S., Meckel, K., Mereghetti, C., Palano, B.: Queue automata of constant length. In: Proceedings of the Descrete Complete Formal Systems, Lecture Notes in Computer Science, vol. 8031, pp. 124–135. Springer (2013)
22. Kutrib, M., Malcher, A., Mereghetti, C., Palano, B.: Descriptional complexity of iterated uniform finite-state transducers. In: Proceedings of the Descrete Complete Formal Systems, Lecture Notes in Computer Science, vol. 11612, pp. 223–234. Springer (2019)
23. Kutrib, M., Malcher, A., Mereghetti, C., Palano, B.: Iterated uniform finite-state transducers: descriptional complexity of nondeterminism and two-way motion. In: Proceedings of the Descrete Complete Formal Systems, Lecture Notes in Computer Science, vol. 12442, pp. 117–129. Springer (2020)
24. Kutrib, M., Malcher, A., Mereghetti, C., Palano, B.: Deterministic and nondeterministic iterated uniform finite-state transducers: computational and descriptional power. In: Proceedings of the Computability in Europe 2020, Lecture Notes in Computer Science, vol. 12098, pp. 87–99. Springer (2020)
25. Kutrib, M., Malcher, A., Mereghetti, C., Palano, B., Wendlandt, M.: Deterministic input-driven queue automata: finite turns, decidability, and closure properties. Theor. Comput. Sci. **578**, 58–71 (2015)
26. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. **30**, 1976–1992 (2001)
27. Mereghetti, C., Palano, B.: Quantum automata for some multiperiodic languages. Theor. Comput. Sci. **387**, 177–186 (2007)
28. Malcher, A., Mereghetti, C., Palano, B.: Descriptional complexity of two-way pushdown automata with restricted head reversals. Theor. Comput. Sci. **449**, 119–133 (2012)
29. Mereghetti, C.: Testing the descriptional power of small Turing machines on nonregular language acceptance. Int. J. Found. Comp. Sci. **19**, 827–843 (2008)
30. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proceedings of the IEEE 12th Symposium Switching on Automation Theory, pp. 188–191 (1971)
31. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3**, 114–125 (1959)