Input-Driven Queue Automata: Finite Turns, Decidability, and Closure Properties

Martin Kutrib[1] Andreas Malcher[1] Carlo Mereghetti[2]
Beatrice Palano[2] Matthias Wendlandt[1]

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
email: {kutrib,malcher,matthias.wendlandt}@informatik.uni-giessen.de

Dipartimento di Informatica, Università degli Studi di Milano
via Comelico 39, 20135 Milano, Italy
email:{mereghetti,palano}@di.unimi.it

Input-driven automata

- The input alphabet is divided into several classes.

Input-driven automata

- The input alphabet is divided into several classes.
- Each class induces a specific behavior of the automaton.

Input-driven automata

- The input alphabet is divided into several classes.
- Each class induces a specific behavior of the automaton.

Example: Input-driven pushdown automata

$$\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$$

Input-driven automata

- The input alphabet is divided into several classes.
- Each class induces a specific behavior of the automaton.

Example: Input-driven pushdown automata

$$\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$$

- $a \in \Sigma_c$: a symbol is pushed onto the pushdown store

Input-driven automata

- The input alphabet is divided into several classes.
- Each class induces a specific behavior of the automaton.

Example: Input-driven pushdown automata

$$\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$$

- $a \in \Sigma_c$: a symbol is pushed onto the pushdown store
- $a \in \Sigma_r$: a symbol is popped from the pushdown store

Input-driven automata

- The input alphabet is divided into several classes.
- Each class induces a specific behavior of the automaton.

Example: Input-driven pushdown automata

$$\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$$

- $a \in \Sigma_c$: a symbol is pushed onto the pushdown store
- $a \in \Sigma_r$: a symbol is popped from the pushdown store
- $a \in \Sigma_i$: internal change of states, no action on the pushdown store

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)
- Visibly pushdown automata (Alur, Madhusudan 2004)

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)
- Visibly pushdown automata (Alur, Madhusudan 2004)
  - nested word automata

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)
- Visibly pushdown automata (Alur, Madhusudan 2004)
  - nested word automata
  - $2^{\Omega(n^2)}$ bounds for determinization

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)
- Visibly pushdown automata (Alur, Madhusudan 2004)
  - nested word automata
  - $2^{\Omega(n^2)}$ bounds for determinization
  - closure properties, decidability questions

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)
- Visibly pushdown automata (Alur, Madhusudan 2004)
  - nested word automata
  - $2^{\Omega(n^2)}$ bounds for determinization
  - closure properties, decidability questions
- Pushdown forest automata (Neumann, Seidl 1998; Gauwin, Niehren, Roos 2008)

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)
- Visibly pushdown automata (Alur, Madhusudan 2004)
  - nested word automata
  - $2^{\Omega(n^2)}$ bounds for determinization
  - closure properties, decidability questions
- Pushdown forest automata (Neumann, Seidl 1998; Gauwin, Niehren, Roos 2008)
- Descriptional complexity aspects (Han, Salomaa 2009, Piao, Salomaa 2009, Okhotin, Salomaa 2011)

Input-driven pushdown automata

- Input-driven PDA accept context-free languages in $O(\log n)$ space. (Mehlhorn 1980, von Braunmühl, Verbeek 1983)
- Nondeterministic and deterministic versions are equivalent. (von Braunmühl, Verbeek 1983)
- Input-driven languages are in NC^1. (Dymond 1988)
- Visibly pushdown automata (Alur, Madhusudan 2004)
  - nested word automata
  - $2^{\Omega(n^2)}$ bounds for determinization
  - closure properties, decidability questions
- Pushdown forest automata (Neumann, Seidl 1998; Gauwin, Niehren, Roos 2008)
- Descriptional complexity aspects (Han, Salomaa 2009, Piao, Salomaa 2009, Okhotin, Salomaa 2011)
- Extensions/generalizations: multiple pushdowns, graph automata, height-deterministic PDA, stacks, . . .

Queue automata

- Introduced by Vollmar in 1970.

Queue automata

- Introduced by Vollmar in 1970.
- General model is equivalent to Turing machines

Queue automata

- Introduced by Vollmar in 1970.
- General model is equivalent to Turing machines
- Restricted real-time, quasi real-time, and linear-time variants.

Queue automata

- Introduced by Vollmar in 1970.
- General model is equivalent to Turing machines
- Restricted real-time, quasi real-time, and linear-time variants.
- Deterministic (DQA) and nondeterministic variants.

Queue automata

- Introduced by Vollmar in 1970.
- General model is equivalent to Turing machines
- Restricted real-time, quasi real-time, and linear-time variants.
- Deterministic (DQA) and nondeterministic variants.
- Extended variants with several queues.

Queue automata

- Introduced by Vollmar in 1970.
- General model is equivalent to Turing machines
- Restricted real-time, quasi real-time, and linear-time variants.
- Deterministic (DQA) and nondeterministic variants.
- Extended variants with several queues.
- Undecidability of emptiness for deterministic queue automata working in real time.

Input-driven queue automata (DVQA)

$M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_e, \delta_r, \delta_i \rangle$,

$$\Sigma = \Sigma_e \cup \Sigma_r \cup \Sigma_i$$

Input-driven queue automata (DVQA)

$M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_e, \delta_r, \delta_i \rangle$,

$$\Sigma = \Sigma_e \cup \Sigma_r \cup \Sigma_i$$

- ➜ $a \in \Sigma_e$: a symbol is stored in the queue

Input-driven queue automata (DVQA)

$M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_e, \delta_r, \delta_i \rangle$,

$$\Sigma = \Sigma_e \cup \Sigma_r \cup \Sigma_i$$

- ➜ $a \in \Sigma_e$: a symbol is stored in the queue
- ➜ $a \in \Sigma_r$: a symbol is removed from the queue

Input-driven queue automata (DVQA)

$M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_e, \delta_r, \delta_i \rangle$,

$$\Sigma = \Sigma_e \cup \Sigma_r \cup \Sigma_i$$

- $a \in \Sigma_e$: a symbol is stored in the queue
- $a \in \Sigma_r$: a symbol is removed from the queue
- $a \in \Sigma_i$: internal change of states, no action on the queue

Input-driven queue automata (DVQA)

$M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_e, \delta_r, \delta_i \rangle$,

$$\Sigma = \Sigma_e \cup \Sigma_r \cup \Sigma_i$$

- $a \in \Sigma_e$: a symbol is stored in the queue
- $a \in \Sigma_r$: a symbol is removed from the queue
- $a \in \Sigma_i$: internal change of states, no action on the queue

By definition, DVQA work in real time.

Example: The language

$$\{\,\$_0\$_1 abb\$_2\$_1 abbabb\$_2\$_1 (abb)^4\$_2 \ldots \$_1 (abb)^{(2^n)}\$_2 \mid n \geq 0\,\}$$

is accepted by the following DVQA.

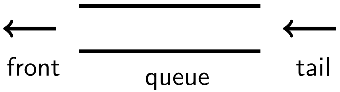
$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$

Example: The language

$$\{\, \$_0\$_1 abb\$_2\$_1 abbabb\$_2\$_1(abb)^4\$_2 \ldots \$_1(abb)^{(2^n)}\$_2 \mid n \geq 0 \,\}$$

is accepted by the following DVQA.

$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$

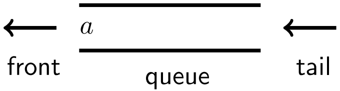


Example: The language

$$\{\, \$_0 \$_1 abb \$_2 \$_1 abbabb \$_2 \$_1 (abb)^4 \$_2 \ldots \$_1 (abb)^{(2^n)} \$_2 \mid n \geq 0 \,\}$$

is accepted by the following DVQA.

$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$

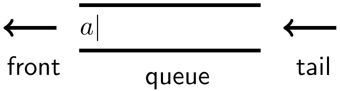


Example: The language

$$\{\, \$_0 \$_1 abb \$_2 \$_1 abbabb \$_2 \$_1 (abb)^4 \$_2 \ldots \$_1 (abb)^{(2^n)} \$_2 \mid n \geq 0 \,\}$$

is accepted by the following DVQA.

$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$

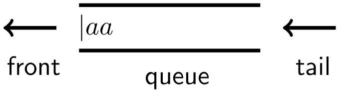


Example: The language

$$\{\,\$_0\$_1 abb\$_2\$_1 abbabb\$_2\$_1(abb)^4\$_2\ldots\$_1(abb)^{(2^n)}\$_2 \mid n \geq 0\,\}$$

is accepted by the following DVQA.

$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$

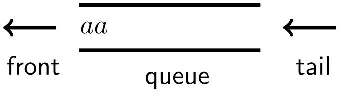


Example: The language

$$\{\, \$_0 \$_1 abb \$_2 \$_1 abbabb \$_2 \$_1 (abb)^4 \$_2 \ldots \$_1 (abb)^{(2^n)} \$_2 \mid n \geq 0 \,\}$$

is accepted by the following DVQA.

$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$

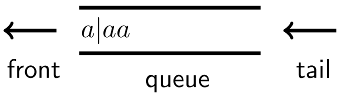


Example: The language

$$\{\, \$_0\$_1abb\$_2\$_1abbabb\$_2\$_1(abb)^4\$_2 \ldots \$_1(abb)^{(2^n)}\$_2 \mid n \geq 0 \,\}$$

is accepted by the following DVQA.

$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$

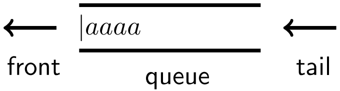


Example: The language

$$\{\, \$_0\$_1 abb\$_2\$_1 abbabb\$_2\$_1(abb)^4\$_2 \ldots \$_1(abb)^{(2^n)}\$_2 \mid n \geq 0 \,\}$$

is accepted by the following DVQA.

$\Sigma_i = \{\}$
$\Sigma_r = \{\$_2, a\}$
$\Sigma_e = \{\$_0, \$_1, b\}$



Turn-bounded queue automata

Turn-bounded queue automata

- For a computation of a queue automaton, a turn is a phase in which the length of the queue first increases and then decreases.

Turn-bounded queue automata

- For a computation of a queue automaton, a turn is a phase in which the length of the queue first increases and then decreases.
- For any given $k \geq 0$, a k-turn computation is any computation containing exactly k turns.

Turn-bounded queue automata

- For a computation of a queue automaton, a turn is a phase in which the length of the queue first increases and then decreases.
- For any given $k \geq 0$, a k-turn computation is any computation containing exactly k turns.
- We restrict deterministic queue automata, to make at most k turns in the queue (DQA_k and DVQA_k).

Flip-pushdown automata

Flip-pushdown automata

- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.

Flip-pushdown automata

- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.
- It has a pushdown store, where letters can be pushed and popped.

Flip-pushdown automata

- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.
- It has a pushdown store, where letters can be pushed and popped.
- Moreover it has the possibility to flip the pushdown store.

Flip-pushdown automata

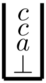
- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.
- It has a pushdown store, where letters can be pushed and popped.
- Moreover it has the possibility to flip the pushdown store.
- Informally a flip means that the pushdown store is reversed.

Flip-pushdown automata

- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.
- It has a pushdown store, where letters can be pushed and popped.
- Moreover it has the possibility to flip the pushdown store.
- Informally a flip means that the pushdown store is reversed.
- In a flip only the pushdown symbol stays on the bottom.

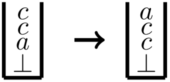
Flip-pushdown automata

- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.
- It has a pushdown store, where letters can be pushed and popped.
- Moreover it has the possibility to flip the pushdown store.
- Informally a flip means that the pushdown store is reversed.
- In a flip only the pushdown symbol stays on the bottom.



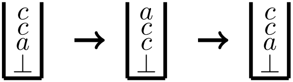
Flip-pushdown automata

- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.
- It has a pushdown store, where letters can be pushed and popped.
- Moreover it has the possibility to flip the pushdown store.
- Informally a flip means that the pushdown store is reversed.
- In a flip only the pushdown symbol stays on the bottom.



Flip-pushdown automata

- The functionality of a deterministic flip-pushdown automaton (DFPDA) [Holzer, Kutrib 2003] is almost the same as of a pushdown automaton.
- It has a pushdown store, where letters can be pushed and popped.
- Moreover it has the possibility to flip the pushdown store.
- Informally a flip means that the pushdown store is reversed.
- In a flip only the pushdown symbol stays on the bottom.



Simulation of turns by flips

Lemma

Let $k \geq 1$ be a constant and M be a k-turn DQA. Then an equivalent $2k$-flip DFPDA can effectively be constructed.

Simulation of turns by flips

Lemma

Let $k \geq 1$ be a constant and M be a k-turn DQA. Then an equivalent $2k$-flip DFPDA can effectively be constructed.

- The idea of the construction is to use one end of the pushdown store as the front and the other end as the tail of the queue.

Simulation of turns by flips

Lemma

Let $k \geq 1$ be a constant and M be a k-turn DQA. Then an equivalent $2k$-flip DFPDA can effectively be constructed.

- The idea of the construction is to use one end of the pushdown store as the front and the other end as the tail of the queue.
- Whenever the queue automaton performs a turn, that is, changes from increasing to decreasing or decreasing to increasing mode, the flip-pushdown automaton flips the front end of the pushdown store to the top.

Flip-pushdown automata

Flip-pushdown automata

- It is shown in [Holzer, Kutrib 2003] that for every DFPDA_k M a context-free language L' that is letter equivalent to $L(M)$ can be constructed.

Flip-pushdown automata

- It is shown in [Holzer, Kutrib 2003] that for every DFPDA_k M a context-free language L' that is letter equivalent to $L(M)$ can be constructed.
- So every language accepted by a queue automaton with a constant number of turns obeys a semilinear Parikh mapping.

Flip-pushdown automata

- It is shown in [Holzer, Kutrib 2003] that for every DFPDA_k M a context-free language L' that is letter equivalent to $L(M)$ can be constructed.
- So every language accepted by a queue automaton with a constant number of turns obeys a semilinear Parikh mapping.

Theorem

Let $k \geq 0$ be a constant and M be a k-turn DQA. Then $L(M)$ is semilinear, in particular, if $L(M)$ is a unary language then it is regular.

Turn hierarchy

Example: Let $h_p : \{a, b\}^* \to \{a', b'\}^*$ be the homomorphism $h_p(a) = a'$, $h_p(b) = b'$. For all $j \geq 0$, we define the sets

$$C_j = \{\, \#w\#h_p(w) \mid w \in \{a, b\}^* \,\}^j \cdot \#$$

and, for all $k \geq 0$ the language $L_k = \bigcup_{j=0}^{k} C_j$.

Turn hierarchy

Example: Let $h_p : \{a, b\}^* \to \{a', b'\}^*$ be the homomorphism $h_p(a) = a'$, $h_p(b) = b'$. For all $j \geq 0$, we define the sets

$$C_j = \{\, \#w\#h_p(w) \mid w \in \{a, b\}^* \,\}^j \cdot \#$$

and, for all $k \geq 0$ the language $L_k = \bigcup_{j=0}^{k} C_j$.

Theorem

Let $k \geq 1$. Then language L_k is accepted by some DVQA_k, but not accepted by any DQA_{k-1}.

Closure properties

	DVQA$_k$	DVQA
$\sim$	no	yes
$\cup_c$	yes	yes
$\cap_c$	yes	yes
$\cap_{REG}$	yes	yes
$\cdot$	no	no
$*$	no	no
h_λ	no	no
h^{-1}	no	no
$\cup$	no	no
$\cap$	no	no

Two signatures
$\Sigma = \Sigma_e \cup \Sigma_r \cup \Sigma_i$
and
$\Sigma' = \Sigma'_e \cup \Sigma'_r \cup \Sigma'_i$
are compatible if
$\bigcup_{j \in \{e,r,i\}} (\Sigma_j \setminus \Sigma'_j) \cap \Sigma' = \emptyset$
and
$\bigcup_{j \in \{e,r,i\}} (\Sigma'_j \setminus \Sigma_j) \cap \Sigma = \emptyset$.

Decidability problems

	DVQA_k	DVQA
emptiness		
finiteness		
universality		
inclusion		
inclusion$_c$		
equivalence		
equivalence$_c$		
finite turn		

- $+$ means *decidable*
- $-$ means *not semidecidable*

Decidability problems

	DVQA_k	DVQA
emptiness	$+$	
finiteness	$+$	
universality		
inclusion		
inclusion$_c$		
equivalence		
equivalence$_c$		
finite turn		

- $+$ means *decidable*
- $-$ means *not semidecidable*

Decidability problems

	DVQA_k	DVQA
emptiness	$+$	
finiteness	$+$	
universality	$+$	
inclusion		
inclusion$_c$		
equivalence		
equivalence$_c$		
finite turn		

- $+$ means *decidable*
- $-$ means *not semidecidable*

Decidability problems

	DVQA_k	DVQA
emptiness	+	
finiteness	+	
universality	+	
inclusion		
inclusion_c	+	
equivalence		
equivalence_c	+	
finite turn		

- $+$ means *decidable*
- $-$ means *not semidecidable*

Decidability problems

Lemma

Let M be an LBA. Then a DVQA accepting $\mathrm{VALC}(M)$ can effectively be constructed.

Decidability problems

	DVQA_k	DVQA
emptiness	$+$	$-$
finiteness	$+$	$-$
universality	$+$	$-$
inclusion		$-$
inclusion$_c$	$+$	$-$
equivalence		$-$
equivalence$_c$	$+$	$-$
finite turn		

- ➜ $+$ means *decidable*
- ➜ $-$ means *not semidecidable*

Decidability problems

	DVQA_k	DVQA
emptiness	$+$	$-$
finiteness	$+$	$-$
universality	$+$	$-$
inclusion	$-$	$-$
inclusion$_c$	$+$	$-$
equivalence		$-$
equivalence$_c$	$+$	$-$
finite turn		

- $+$ means *decidable*
- $-$ means *not semidecidable*

Decidability problems

	DVQA_k	DVQA
emptiness	$+$	$-$
finiteness	$+$	$-$
universality	$+$	$-$
inclusion	$-$	$-$
inclusion$_c$	$+$	$-$
equivalence		$-$
equivalence$_c$	$+$	$-$
finite turn	trivial	$-$

- $+$ means *decidable*
- $-$ means *not semidecidable*

Decidability problems

	DVQA_k	DVQA
emptiness	$+$	$-$
finiteness	$+$	$-$
universality	$+$	$-$
inclusion	$-$	$-$
inclusion$_c$	$+$	$-$
equivalence	?	$-$
equivalence$_c$	$+$	$-$
finite turn	trivial	$-$

- $+$ means *decidable*
- $-$ means *not semidecidable*