Queue Automata of Constant Length

Sebastian Jakobi Katja Meckel
Carlo Mereghetti Beatrice Palano

Institut für Informatik, Universität Giessen, Germany
Dipartimento di Informatica, Università degli Studi di Milano
Milano, Italy

Theorietag 2013, Ilmenau

Finite State Automata with Memory

Starting from DFA and NFA we can define

1. Pushdown Automata (PDA) by adding a LIFO storage
2. Queue Automata (QA) by adding a FIFO storage

Note

These automata models are simply Turing machines with restrictions of the access of the working tape.

Finite State Automata with Memory

Starting from DFA and NFA we can define

1. Pushdown Automata (PDA) by adding a LIFO storage
2. Queue Automata (QA) by adding a FIFO storage

Note

These automata models are simply Turing machines with restrictions of the access of the working tape.

Restricting the storage sizes to a constant h we get

1. constant height PDA (h-PDA)
2. constant length QA (h-QA)

Measuring Constant Memory Automata

Measure for Constant Memory Automata

Our measure for constant memory automata is an ordered triple consisting of

1. the number of states of the finite control,
2. the size of memory alphabet,
3. the memory limit.

This definition was already used for results on constant height pushdown automata. [Geffert, CM, BP 2010]

Conversion h-PDA $\rightarrow$ FA

The following trade-offs were shown
[Bednárová, Geffert, CM, BP 2012]

- h-NPDA $\rightarrow$ NFA: exponential
- h-NPDA $\rightarrow$ DFA: double exponential

Conversion h-PDA $\rightarrow$ FA

The following trade-offs were shown
[Bednárová, Geffert, CM, BP 2012]

- h-NPDA $\rightarrow$ NFA: exponential
- h-NPDA $\rightarrow$ DFA: double exponential
- h-DPDA $\rightarrow$ NFA: exponential
- h-DPDA $\rightarrow$ DFA: exponential

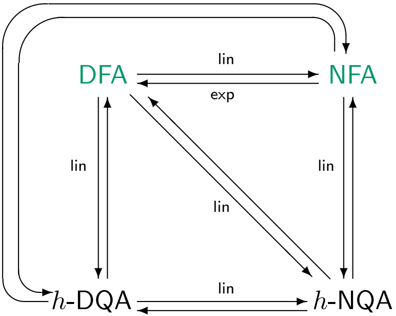
Conversion h-PDA $\rightarrow$ FA

The following trade-offs were shown
[Bednárová, Geffert, CM, BP 2012]

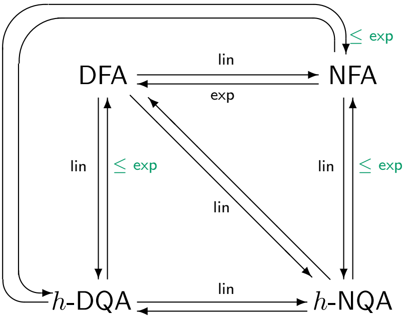
- h-NPDA $\rightarrow$ NFA: exponential
- h-NPDA $\rightarrow$ DFA: double exponential
- h-DPDA $\rightarrow$ NFA: exponential
- h-DPDA $\rightarrow$ DFA: exponential

What are the trade-offs for the conversions between constant length queue automata and finite automata?

Conversion h-QA $\rightarrow$ FA



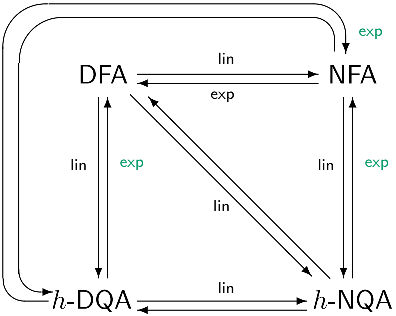
Conversion h-NQA $\to$ NFA



Theorem

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 $|Q'| \leq |Q| \cdot \left|\Gamma^{\leq h}\right|$. Moreover, if A is a DQA then A' is a DFA.

Conversion h-NQA $\to$ NFA



Optimality of the bound

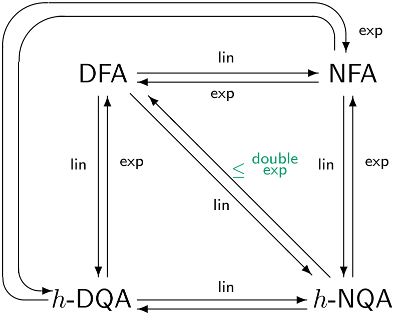
The language

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

is accepted by

- a constant length DQA with 3 states, queue alphabet $\Gamma \cup \{\#, \vdash\}$, and constant length h,
- an NFA with at least $\left|\Gamma^{\leq h}\right|$ states (fooling set).

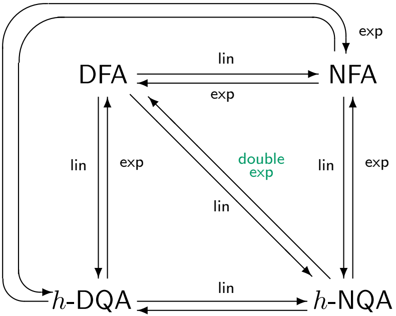
Conversion h-NQA $\to$ DFA



Theorem

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

Conversion h-NQA $\to$ DFA

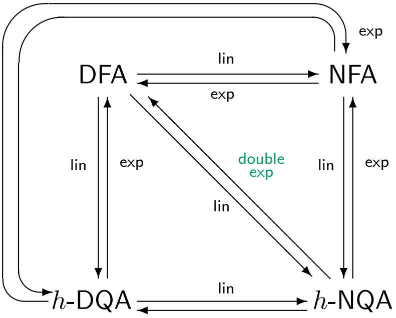


Optimality of the bound

The language

$$S_{\Gamma,h} = \{v_1 v_2 \dots v_r \# w_1 w_2 \dots w_t \mid v_i, w_j \in \Gamma^h, \exists i, j : v_i = w_j\}$$

Conversion h-NQA $\to$ DFA

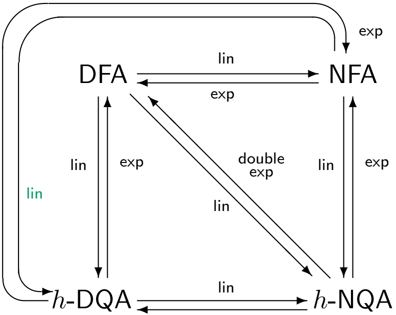


Optimality of the bound

is accepted by

- a constant length NQA with $O(h)$ states, queue alphabet $\Gamma \cup \{\vdash\}$, and constant length h,
- a DFA with at least $2^{|\Gamma^h|}$ states.

Conversion NFA $\to$ h-DQA



Theorem

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 $|Q'| \in O(|Q| \cdot |\Sigma|)$, and $|\Gamma|, h \in O(|Q|)$.

Conversion NFA $\to$ h-DQA

Key idea of the proof

- In its queue the constant length DQA stores the set of successor states the NFA A may be in on reading an input symbol.
- Let $\delta(s_1, a) = \{r_1, \ldots, r_k\}$ be a transition of A and the queue content of A' be $s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash$ the DQA converts its queue content in the following way by using λ-moves:

$$\begin{array}{ll} & s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\ \stackrel{a}{\to} & s_2 \ldots s_l \# t_1 \ldots t_m \vdash \end{array}$$

Conversion NFA $\to$ h-DQA

Key idea of the proof

- In its queue the constant length DQA stores the set of successor states the NFA A may be in on reading an input symbol.
- Let $\delta(s_1, a) = \{r_1, \ldots, r_k\}$ be a transition of A and the queue content of A' be $s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash$ the DQA converts its queue content in the following way by using λ-moves:

$$
\begin{aligned}
& s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{a}{\to} \quad & s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{\lambda}{\to} \cdots \stackrel{\lambda}{\to} \quad & \# t_1 \ldots t_m \vdash s_2 \ldots s_l
\end{aligned}
$$

Conversion NFA $\rightarrow$ h-DQA

Key idea of the proof

- In its queue the constant length DQA stores the set of successor states the NFA A may be in on reading an input symbol.
- Let $\delta(s_1, a) = \{r_1, \ldots, r_k\}$ be a transition of A and the queue content of A' be $s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash$ the DQA converts its queue content in the following way by using λ-moves:

$$
\begin{aligned}
& s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{a}{\rightarrow}\ & s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{\lambda}{\rightarrow} \cdots \stackrel{\lambda}{\rightarrow}\ & \# t_1 \ldots t_m \vdash s_2 \ldots s_l \\
\stackrel{\lambda}{\rightarrow}\ & t_1 \ldots t_m \vdash s_2 \ldots s_l \#
\end{aligned}
$$

Conversion NFA $\to$ h-DQA

Key idea of the proof

- In its queue the constant length DQA stores the set of successor states the NFA A may be in on reading an input symbol.
- Let $\delta(s_1, a) = \{r_1, \ldots, r_k\}$ be a transition of A and the queue content of A' be $s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash$ the DQA converts its queue content in the following way by using λ-moves:

$$
\begin{aligned}
& s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{a}{\to}\ & s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{\lambda}{\to} \cdots \stackrel{\lambda}{\to}\ & \# t_1 \ldots t_m \vdash s_2 \ldots s_l \\
\stackrel{\lambda}{\to}\ & t_1 \ldots t_m \vdash s_2 \ldots s_l \# \\
\stackrel{\lambda}{\to} \cdots \stackrel{\lambda}{\to}\ & \vdash s_2 \ldots s_l \# t_1 \ldots t_{i-1} t_{i+1} \ldots t_m
\end{aligned}
$$

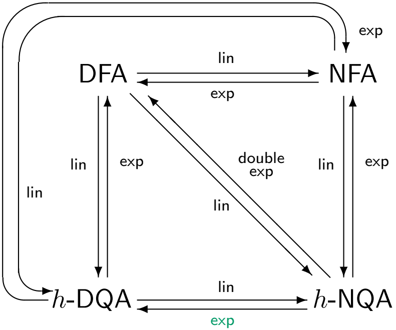
Conversion NFA $\rightarrow$ h-DQA

Key idea of the proof

- In its queue the constant length DQA stores the set of successor states the NFA A may be in on reading an input symbol.
- Let $\delta(s_1, a) = \{r_1, \ldots, r_k\}$ be a transition of A and the queue content of A' be $s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash$ the DQA converts its queue content in the following way by using λ-moves:

$$
\begin{array}{rl}
& s_1 s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{a}{\rightarrow} & s_2 \ldots s_l \# t_1 \ldots t_m \vdash \\
\stackrel{\lambda}{\rightarrow} \cdots \stackrel{\lambda}{\rightarrow} & \# t_1 \ldots t_m \vdash s_2 \ldots s_l \\
\stackrel{\lambda}{\rightarrow} & t_1 \ldots t_m \vdash s_2 \ldots s_l \# \\
\stackrel{\lambda}{\rightarrow} \cdots \stackrel{\lambda}{\rightarrow} & \vdash s_2 \ldots s_l \# t_1 \ldots t_{i-1} t_{i+1} \ldots t_m \\
\stackrel{\lambda}{\rightarrow} & s_2 \ldots s_l \# t_1 \ldots t_{i-1} t_{i+1} \ldots t_m r_1 \ldots r_k \vdash
\end{array}
$$

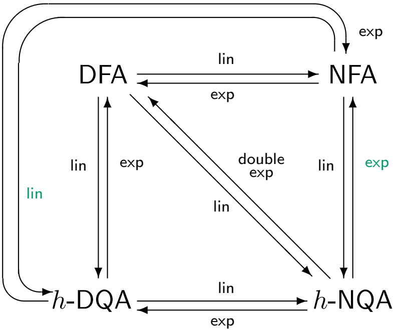
Determinization of h-NQA



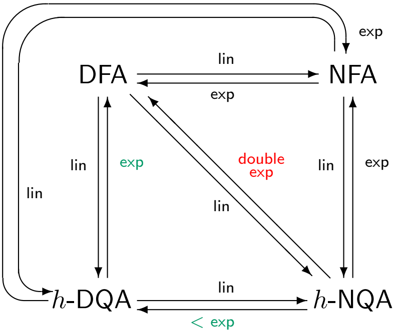
Theorem

For each constant length NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h \rangle$ there exists an equivalent constant length DQA $A' = \langle Q', \Sigma, \Gamma', \delta', q'_0, \vdash', F', h' \rangle$ with $|Q'| \in O(|Q| \cdot \left|\Gamma^{\leq h}\right| \cdot |\Sigma|)$ and $|\Gamma'|, h' \in O(|Q| \cdot \left|\Gamma^{\leq h}\right|)$. Furthermore, this conversion is optimal.

Determinization of h-NQA

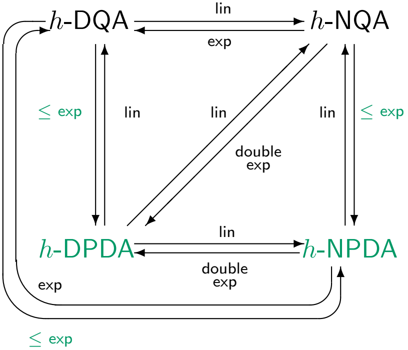


Determinization of h-NQA



Conversions between h-PDA and h-QA

Conversions between h-PDA and h-QA



Straight Line Programs for Regular Expressions

Straight line programs (SLP) are used for the representation of regular expressions. Given an alphabet Σ and a set of variables $X = \{x_1, \ldots, x_l\}$, an SLP is defined to be a finite sequence of instructions of the form

- $x_i := \emptyset, x_i := \lambda$, or $x_i := a$ for any $a \in \Sigma$
- $x_i := x_j + x_k, x_i := x_j \cdot x_k$, or $x_i := x_j^*$ for $1 \leq j, k < i$

An SLP is always loopless. The variable x_l contains the regular expression.

Straight Line Programs for Regular Expressions

Straight line programs (SLP) are used for the representation of regular expressions. Given an alphabet Σ and a set of variables $X = \{x_1, \ldots, x_l\}$, an SLP is defined to be a finite sequence of instructions of the form

- $x_i := \emptyset, x_i := \lambda$, or $x_i := a$ for any $a \in \Sigma$
- $x_i := x_j + x_k, x_i := x_j \cdot x_k$, or $x_i := x_j^*$ for $1 \leq j, k < i$

An SLP is always loopless. The variable x_l contains the regular expression.

Measure for an SLP P: (length(P),fan-out(P))

- length(P): number of instructions in P
- fan-out(P): maximum of occurrences of a reused variable

Conversion h-NPDA $\rightarrow$ SLP

Theorem [Geffert, CM, BP 2010]

Let $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \{q_f\}, \vdash, h\rangle$ be a constant height NPDA. Then there exists an SLP P_A such that $reg\text{-}exp(P)$ denotes $L(A)$, with length$(P_A) \leq O(h \cdot |Q|^4 \cdot |\Gamma| + |Q|^2 \cdot |\Sigma|)$ and fan-out$(P_A) \leq |Q|^2 + 1$.

Conversion h-NQA $\to$ SLP

Theorem

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

Conversion h-NQA $\to$ SLP

The optimality of this bound is proved by the language

$$L_{\Gamma,h} = \bigcup_{u \in \Gamma^h} \{(\#u)^i\$: i \geq 1\}$$

Conversion h-NQA $\to$ SLP

$$L_{\Gamma,h} = \bigcup_{u \in \Gamma^h} \{(\#u)^i \$: i \geq 1\}$$

$L_{\Gamma,h}$ is accepted by

- ➜ a constant length DQA with $O(h)$ states, queue alphabet $\Gamma \cup \{\vdash, \#\}$, and queue length $h + 1$.
- ➜ an SLP with at least $\left|\Gamma^h\right|$ variables.

Conversion h-NQA $\to$ SLP

$$L_{\Gamma,h} = \bigcup_{u \in \Gamma^h} \{(\#u)^i \$: i \geq 1\}$$

- x is called a star-variable if it occurs in a star-instruction $x := y^*$. $L(x)$ denotes the language represented by the regular expression computed in x.

Conversion h-NQA $\rightarrow$ SLP

$$L_{\Gamma,h} = \bigcup_{u \in \Gamma^h} \{(\#u)^i\$: i \geq 1\}$$

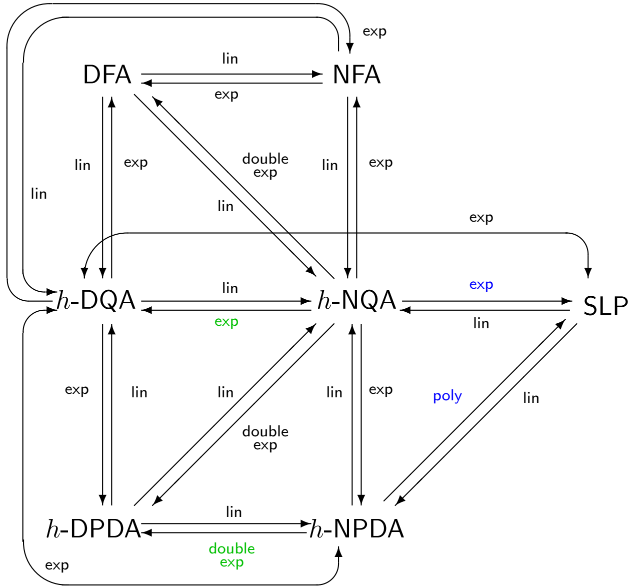
- Let the SLP P compute a regular expression describing $L_{\Gamma,h}$.
- The SLP P' is obtained from P by replacing every instruction $x := y^*$ by $x := \lambda$.
- P' describes a finite language. Let m denote the length of its longest word.
- For all words $z \in L_{\Gamma,h}$ with $|z| > m$, P must use a star-variable to produce it.

Conversion h-NQA $\rightarrow$ SLP

$$L_{\Gamma,h} = \bigcup_{u \in \Gamma^h} \{(\#u)^i\$: i \geq 1\}$$

- Choosing $z_u = (\#u)^m\$ \in L_{\Gamma,h}, u \in \Gamma^h$, we know that there exists a star-variable x_u in P that produces one part of z_u.
- We can show that there exists a star-variable x_u such that $L(x_u)$ contains a word having the factor $\#u\#$.
- If P has less than Γ^h variables, there exist $u, v \in \Gamma^h, u \neq v$, such that $x_u = x_v$.
- Then P describes words of the form $\alpha\#u\#\beta\#v\#\gamma \notin L_{\Gamma,h}$.

Summary of the Results



Thank you for your attention