

Brzozowski Goes Concurrent

A Kleene Theorem for Pomset Languages

*Tobias Kappé*¹ Paul Brunet¹ Bas Luttik² Alexandra Silva¹ Fabio Zanasi¹

¹University College London

²Eindhoven University of Technology

CONCUR 2017

Introduction

Kleene Algebra models *program flow*.

- abort (0) and skip (1)
- non-deterministic choice (+)
- sequential composition ($\cdot$)
- indefinite repetition ($*$)

Introduction

Kleene Algebra models *program flow*.

- abort (0) and skip (1)
- non-deterministic choice (+)
- sequential composition ($\cdot$)
- indefinite repetition ($*$)

Thread 1	Thread 2
$x \leftarrow 1$	$y \leftarrow 1$
$r_1 \leftarrow y$	$r_2 \leftarrow x$

Introduction

Kleene Algebra models *program flow*.

- abort (0) and skip (1)
- non-deterministic choice (+)
- sequential composition ($\cdot$)
- indefinite repetition ($*$)
- parallel composition ($\parallel$)

Thread 1	Thread 2
$x \leftarrow 1$	$y \leftarrow 1$
$r_1 \leftarrow y$	$r_2 \leftarrow x$

Introduction

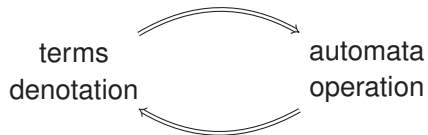
Kleene Algebra models *program flow*.

- abort (0) and skip (1)
- non-deterministic choice (+)
- sequential composition ($\cdot$)
- indefinite repetition ($*$)
- parallel composition ($\parallel$)

Thread 1	Thread 2
$x \leftarrow 1$	$y \leftarrow 1$
$r_1 \leftarrow y$	$r_2 \leftarrow x$

$(x \leftarrow 1; r_1 \leftarrow y) \parallel (y \leftarrow 1; r_2 \leftarrow x)$

Kleene theorem:



Existing Kleene Theorems¹ have ...

- ... non-deterministic automata
- ... a *semantic* precondition on automata

¹Jipsen 2014; Lodaya and Weil 2000.

Existing Kleene Theorems¹ have ...

- ... non-deterministic automata
- ... a *semantic* precondition on automata

Our Kleene Theorem has ...

- ... (sequentially) deterministic automata
- ... a *syntactic* precondition on automata

¹Jipsen 2014; Lodaya and Weil 2000.

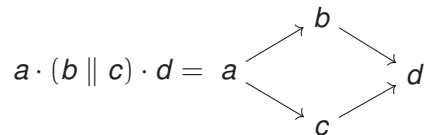
$\mathcal{T}$ given by the grammar

$$e, f ::= 0 \mid 1 \mid a \in \Sigma \mid e + f \mid e \cdot f \mid e \parallel f \mid e^*$$

$\mathcal{T}$ given by the grammar

$$e, f ::= 0 \mid 1 \mid a \in \Sigma \mid e + f \mid e \cdot f \mid e \parallel f \mid e^*$$

Preliminaries



$$\llbracket - \rrbracket : \mathcal{T} \rightarrow 2^{\text{Pom}_\Sigma}$$

$$\llbracket 0 \rrbracket = \emptyset$$

$$\llbracket 1 \rrbracket = \{1\}$$

$$\llbracket a \rrbracket = \{a\}$$

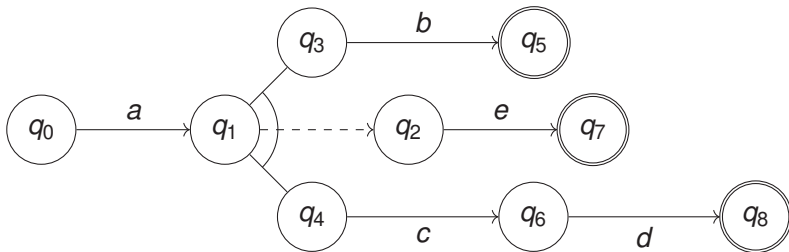
$$\llbracket e + f \rrbracket = \llbracket e \rrbracket \cup \llbracket f \rrbracket$$

$$\llbracket e \cdot f \rrbracket = \llbracket e \rrbracket \cdot \llbracket f \rrbracket$$

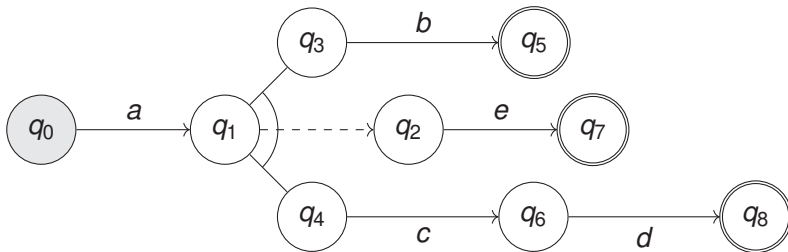
$$\llbracket e \parallel f \rrbracket = \llbracket e \rrbracket \parallel \llbracket f \rrbracket$$

$$\llbracket e^* \rrbracket = \llbracket e \rrbracket^*$$

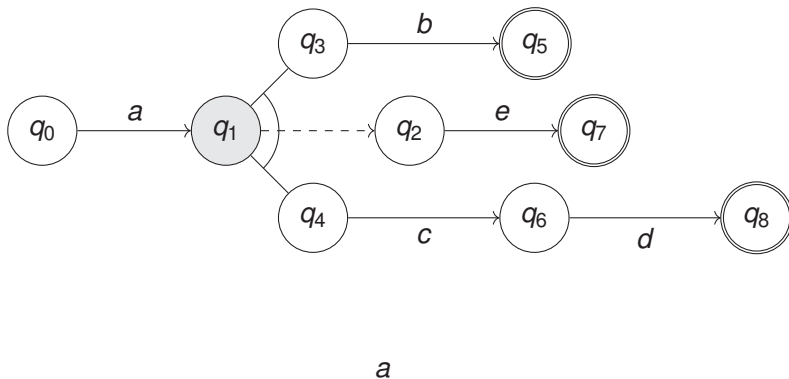
Pomset Automata



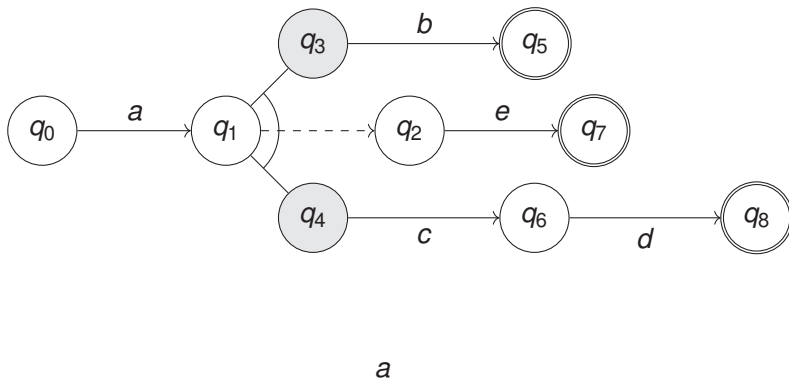
Pomset Automata



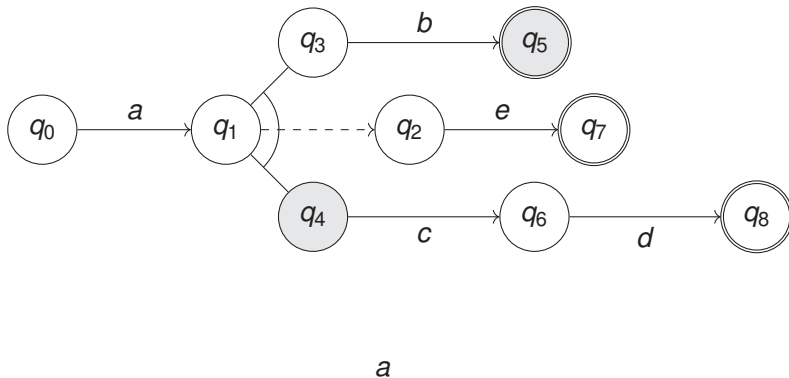
Pomset Automata



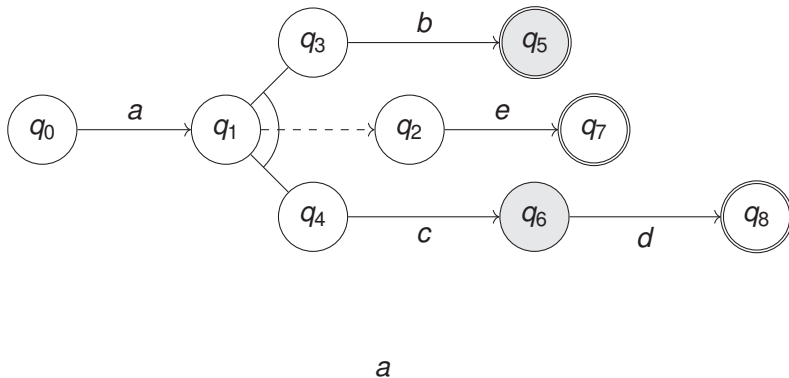
Pomset Automata



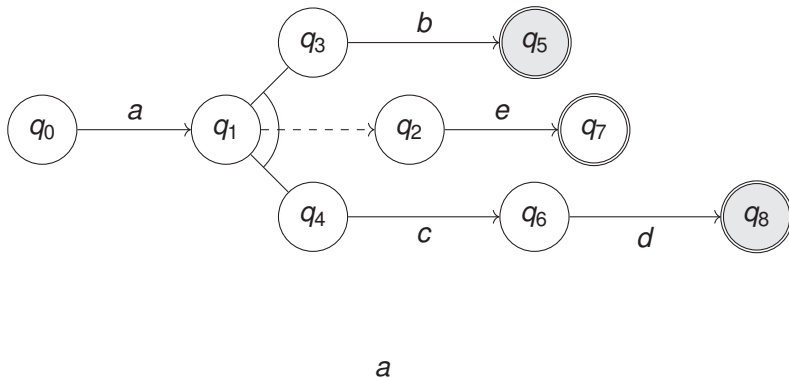
Pomset Automata



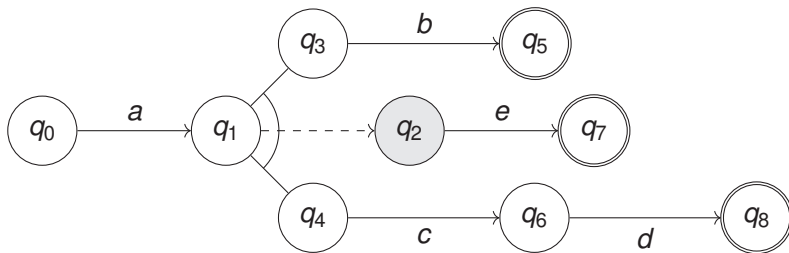
Pomset Automata



Pomset Automata

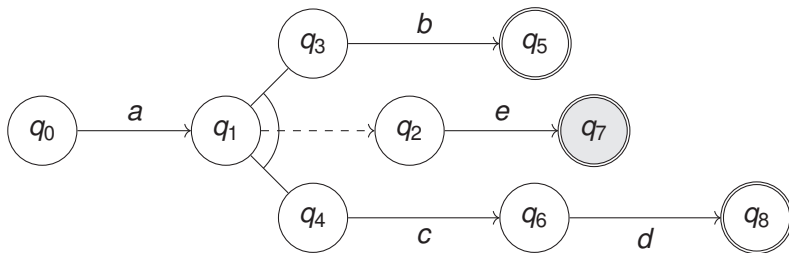


Pomset Automata



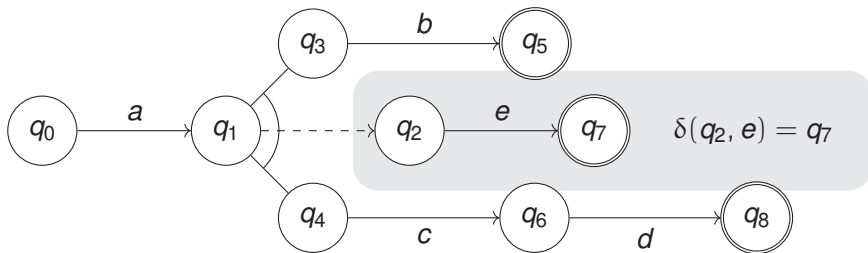
$a(b \parallel cd)$

Pomset Automata

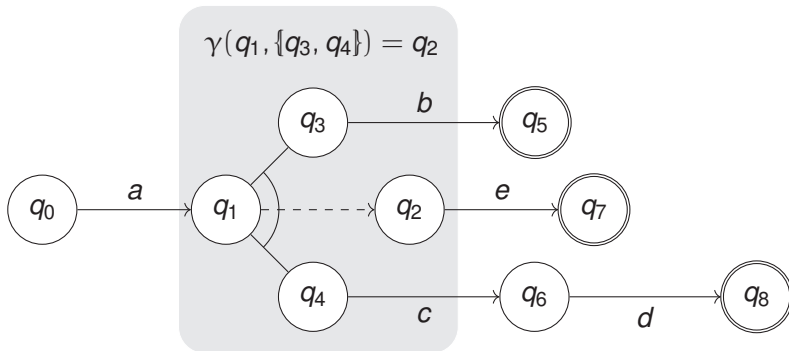


$a(b \parallel cd)e$

Pomset Automata



Pomset Automata



Definition

A *pomset automaton (PA)* is a tuple $\langle Q, \delta, \gamma, F \rangle$ with

- Q a set of *states*, $F \subseteq Q$ the *accepting states*,
- $\delta : Q \times \Sigma \rightarrow Q$, the *sequential transition function*,
- $\gamma : Q \times \mathcal{M}_\omega(Q) \rightarrow Q$, a the *parallel transition function*

Definition

$\rightarrow_A \subseteq Q \times \text{Pom}_\Sigma \times Q$ is the smallest relation such that

$$\begin{array}{c} \frac{}{q \xrightarrow{a}_A \delta(q, a)} \qquad \frac{q \xrightarrow{U}_A q'' \quad q'' \xrightarrow{V}_A q'}{q \xrightarrow{U \cdot V}_A q'} \qquad \frac{r \xrightarrow{U}_A r' \in F \quad s \xrightarrow{V}_A s' \in F}{q \xrightarrow{U \parallel V}_A \gamma(q, \{r, s\})} \end{array}$$

Definition

$\rightarrow_A \subseteq Q \times \text{Pom}_\Sigma \times Q$ is the smallest relation such that

$$\frac{}{q \xrightarrow{a}_A \delta(q, a)} \quad \frac{q \xrightarrow{U}_A q'' \quad q'' \xrightarrow{V}_A q'}{q \xrightarrow{U \cdot V}_A q'} \quad \frac{r \xrightarrow{U}_A r' \in F \quad s \xrightarrow{V}_A s' \in F}{q \xrightarrow{U \parallel V}_A \gamma(q, \{r, s\})}$$

$L_A(q)$ is defined by

$$L_A(q) = \{U : q \xrightarrow{U}_A q' \in F\} \cup \{1 : q \in F\}$$

Expressions to Automata

Brzozowski-construction:

1 $\mathcal{T}$ as state space

Expressions to Automata

Brzozowski-construction:

- 1 $\mathcal{T}$ as state space
- 2 $F_\Sigma \subseteq \mathcal{T}$: accepting expressions

Expressions to Automata

Brzozowski-construction:

- 1 $\mathcal{T}$ as state space
- 2 $F_\Sigma \subseteq \mathcal{T}$: accepting expressions
- 3 $\delta_\Sigma : \mathcal{T} \times \Sigma \rightarrow \mathcal{T}$: sequential derivatives

Expressions to Automata

Brzowski-construction:

- 1 $\mathcal{T}$ as state space
- 2 $F_\Sigma \subseteq \mathcal{T}$: accepting expressions
- 3 $\delta_\Sigma : \mathcal{T} \times \Sigma \rightarrow \mathcal{T}$: sequential derivatives
- 4 $\gamma_\Sigma : \mathcal{T} \times \mathcal{M}_\omega(\mathcal{T}) \rightarrow \mathcal{T}$: parallel derivatives

Expressions to Automata

Brzowski-construction:

- 1 $\mathcal{T}$ as state space
- 2 $F_\Sigma \subseteq \mathcal{T}$: accepting expressions
- 3 $\delta_\Sigma : \mathcal{T} \times \Sigma \rightarrow \mathcal{T}$: sequential derivatives
- 4 $\gamma_\Sigma : \mathcal{T} \times \mathcal{M}_\omega(\mathcal{T}) \rightarrow \mathcal{T}$: parallel derivatives
- 5 trim to finite automaton as necessary

Expressions to Automata

Brzozowski-construction:

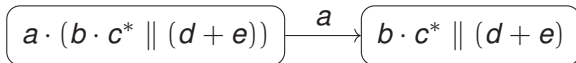
- 1 $\mathcal{T}$ as state space
- 2 $F_\Sigma \subseteq \mathcal{T}$: accepting expressions
- 3 $\delta_\Sigma : \mathcal{T} \times \Sigma \rightarrow \mathcal{T}$: sequential derivatives
- 4 $\gamma_\Sigma : \mathcal{T} \times \mathcal{M}_\omega(\mathcal{T}) \rightarrow \mathcal{T}$: parallel derivatives
- 5 trim to finite automaton as necessary

We write $L_\Sigma(e)$ for $L_{A_\Sigma}(e)$.

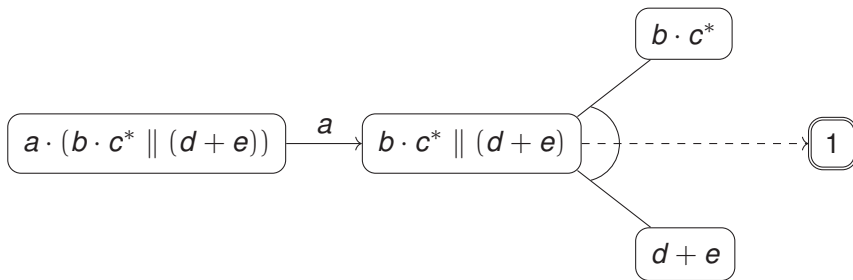
Expressions to Automata

$$a \cdot (b \cdot c^* \parallel (d + e))$$

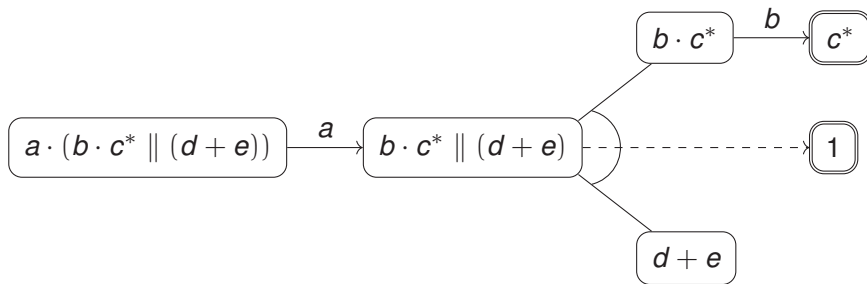
Expressions to Automata



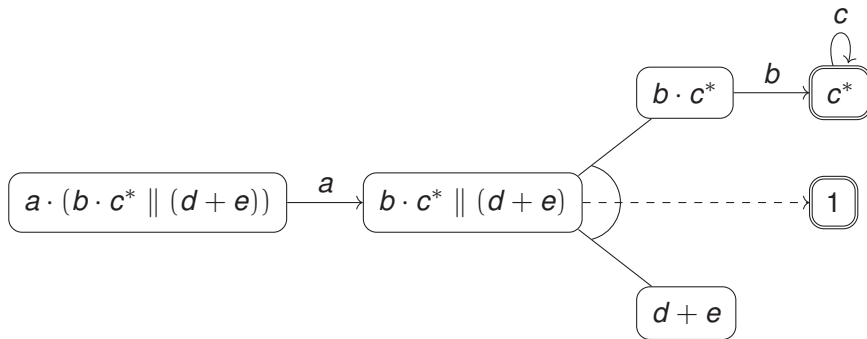
Expressions to Automata



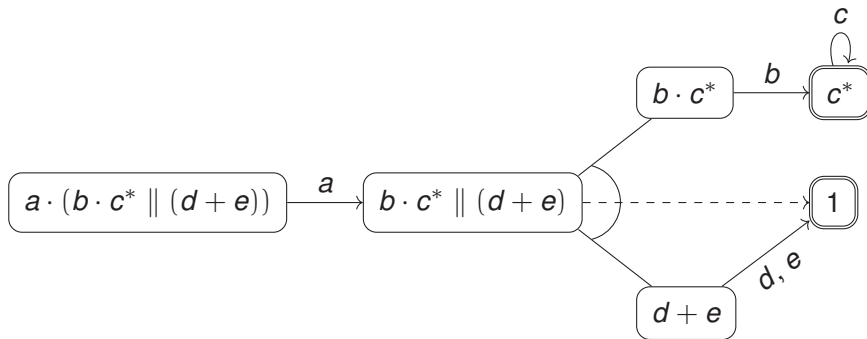
Expressions to Automata



Expressions to Automata



Expressions to Automata



Expressions to Automata

Theorem

Let $e \in \mathcal{T}$; then $L_{\Sigma}(e) = \llbracket e \rrbracket$.

Expressions to Automata

Theorem

Let $e \in \mathcal{T}$; then $L_{\Sigma}(e) = \llbracket e \rrbracket$.

Proof in two parts:

Expressions to Automata

Theorem

Let $e \in \mathcal{T}$; then $L_{\Sigma}(e) = \llbracket e \rrbracket$.

Proof in two parts:

- Deconstruction: if $e \xrightarrow{U}_{\gg \Sigma} f$, then ...

Expressions to Automata

Theorem

Let $e \in \mathcal{T}$; then $L_{\Sigma}(e) = \llbracket e \rrbracket$.

Proof in two parts:

- Deconstruction: if $e \xrightarrow{U}_{\Sigma} f$, then ...
- Construction: if ..., then $e \xrightarrow{U}_{\Sigma} f$.

Expressions to Automata

Theorem

Let $e \in \mathcal{T}$; then $L_\Sigma(e) = \llbracket e \rrbracket$.

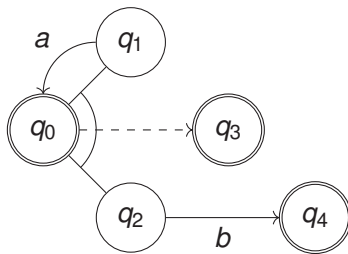
Proof in two parts:

- Deconstruction: if $e \xrightarrow{U}_{\gg \Sigma} f$, then ...
- Construction: if ..., then $e \xrightarrow{U}_{\gg \Sigma} f$.

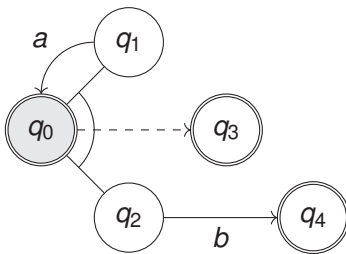
Theorem

Let $e \in \mathcal{T}$; we find a finite A with a state q such that $L_A(q) = L_\Sigma(e)$.

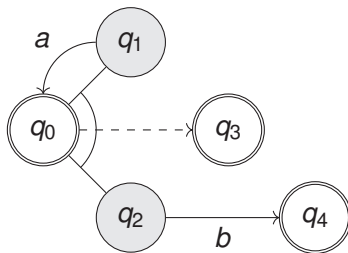
Automata to Expressions



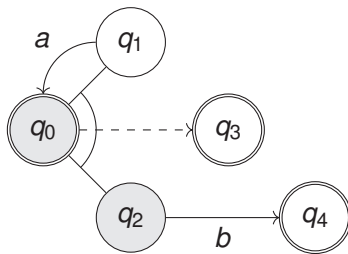
Automata to Expressions



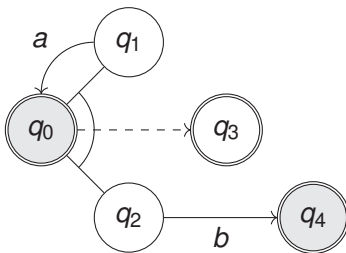
Automata to Expressions



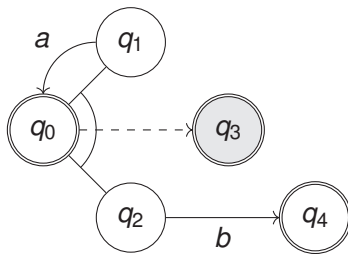
Automata to Expressions



Automata to Expressions

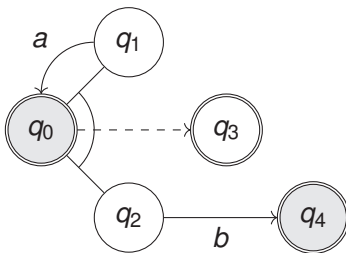


Automata to Expressions

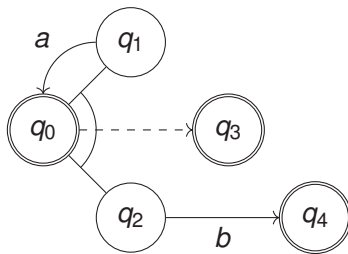


$a \parallel b$

Automata to Expressions

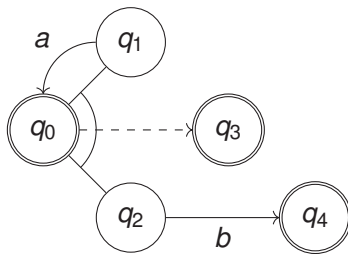


Automata to Expressions



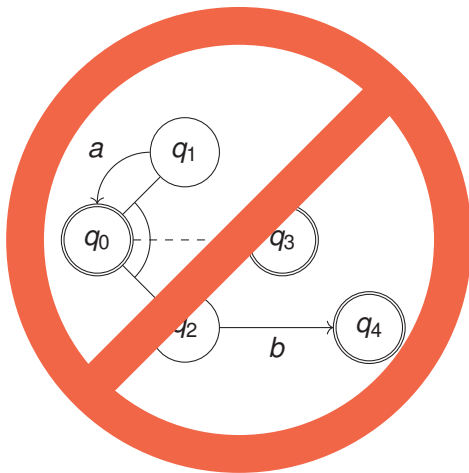
$$a(a \parallel b) \parallel b$$

Automata to Expressions

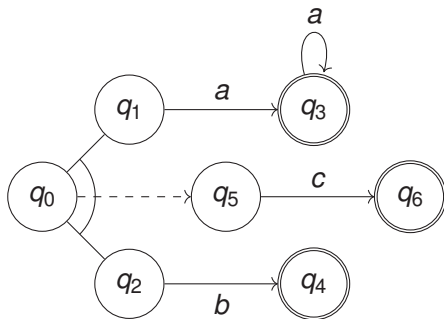


$$a(a(a \parallel b)) \parallel b$$

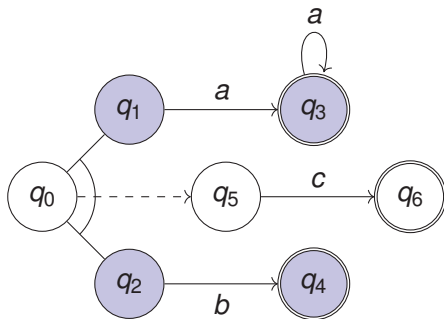
Automata to Expressions



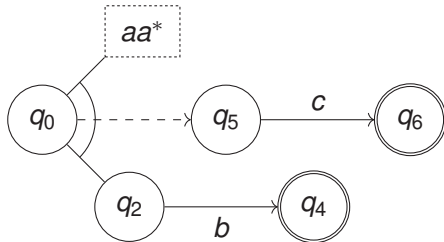
Automata to Expressions



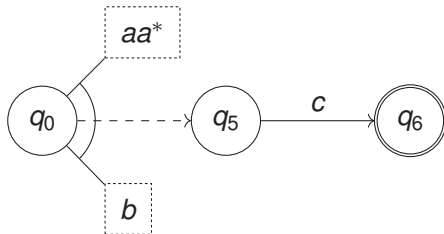
Automata to Expressions



Automata to Expressions



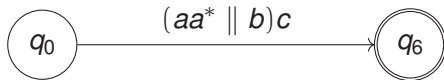
Automata to Expressions



Automata to Expressions



Automata to Expressions



Further Work

Extend to WCKA: exchange law.

²Anderson et al. 2014.

Further Work

Extend to WCKA: exchange law.

Possible applications

- Completeness proof based on automata.
- Equivalence checking of WBKA expressions.

²Anderson et al. 2014.

Further Work

Extend to WCKA: exchange law.

Possible applications

- Completeness proof based on automata.
- Equivalence checking of WBKA expressions.

Endgame: concurrent extension of NetKAT.²

²Anderson et al. 2014.