Brzozowski Goes Concurrent

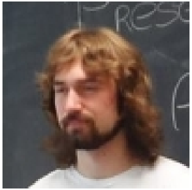
A Kleene Theorem for Pomset Languages

FSCD 2017

CoNeCo

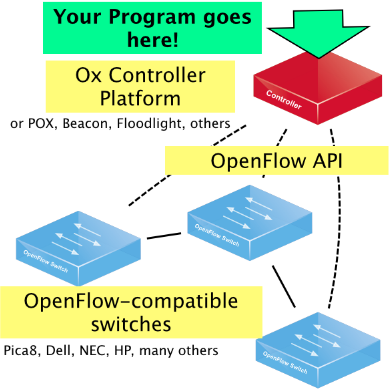
Concurrency, Networks, and Coinduction

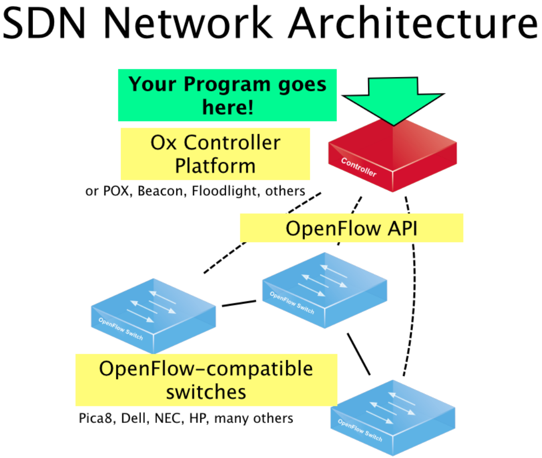
Introduction



Tobias Kappe Paul Brunet Bas Luttik Fabio Zanasi

SDN Network Architecture





Goals of new network PL:

- raise the level of abstraction above hardware-based APIs (OpenFlow)
- make it easier to build sophisticated and reliable SDN applications and reason about them

Introduction

NetKAT

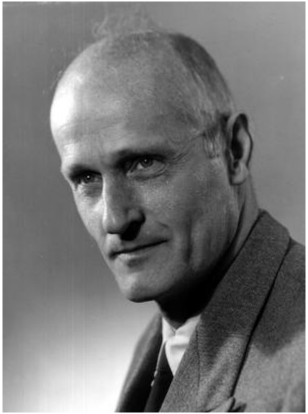
=

Kleene algebra with tests (KAT)

+

additional specialized constructs particular to
network topology and packet switching

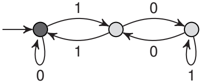
Introduction



Stephen Cole Kleene
(1909–1994)

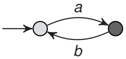
$(0 + 1(01^*0)^*1)^*$

{multiples of 3 in binary}



$(ab)^*a = a(ba)^*$

$\{a, aba, ababa, \ldots\}$



$(a + b)^* = a^*(ba^*)^*$

{all strings over $\{a, b\}$}

Introduction

$(K, B, +, \cdot, ^*, \bar{\ }, 0, 1), \quad B \subseteq K$

- $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra
- $(B, +, \cdot, \bar{\ }, 0, 1)$ is a Boolean algebra
- $(B, +, \cdot, 0, 1)$ is a subalgebra of $(K, +, \cdot, 0, 1)$

- $p, q, r, \ldots$ range over K
- $a, b, c, \ldots$ range over B

Introduction

$$\begin{aligned} p; q &\triangleq pq \\ \textbf{if } b \textbf{ then } p \textbf{ else } q &\triangleq bp + \bar{b}q \\ \textbf{while } b \textbf{ do } p &\triangleq (bp)^* \bar{b} \end{aligned}$$

Introduction

$$
\begin{aligned}
\{b\}\, p\, \{c\} \quad &\overset{\triangle}{\iff} \quad bp \leqslant pc \\
&\iff \quad bp = bpc \\
&\iff \quad bp\bar{c} = 0
\end{aligned}
$$

The Hoare while rule

$$\frac{\{bc\}\, p\, \{c\}}{\{c\}\, \textbf{while}\ b\ \textbf{do}\ p\, \{\bar{b}c\}}$$

becomes the universal Horn sentence

$$bcp\bar{c} = 0 \;\; \Rightarrow \;\; c(bp)^*\bar{b}\;\bar{\bar{b}} = 0$$

Introduction

Deductive Completeness and Complexity

- deductively complete over language, relational, and trace models
- subsumes propositional Hoare logic (PHL)
- deductively complete for all relationally valid Hoare-style rules

$$\frac{\{b_1\}\, p_1\, \{c_1\}, \ \ldots, \ \{b_n\}\, p_n\, \{c_n\}}{\{b\}\, p\, \{c\}}$$

- decidable in PSPACE

Applications

- protocol verification
- static analysis and abstract interpretation
- verification of compiler optimizations

NetKAT (POPL'14, POPL'15, ICFP'15, . . .)

Introduction

- a *packet* π is an assignment of constant values n to fields x
- a *packet history* is a nonempty sequence of packets $\pi_1 :: \pi_2 :: \cdots :: \pi_k$
- the *head packet* is π_1

NetKAT

- assignments $x \leftarrow n$
  assign constant value n to field x in the head packet
- tests $x = n$
  if value of field x in the head packet is n, then pass, else drop

Example

$$sw = 6 \,;\, pt = 88 \,;\, dest \leftarrow 10.0.0.1 \,;\, pt \leftarrow 50$$

"For all packets incoming on port 88 of switch 6, set the destination IP address to 10.0.0.1 and send the packet out on port 50."

Introduction

Reachability

- Can host A communicate with host B? Can every host communicate with every other host?

Security

- Does all untrusted traffic pass through the intrusion detection system located at C?

Loop detection

- Is it possible for a packet to be forwarded around a cycle in the network?

Probabilistic NetKAT (ESOP' 16, POPL' 17)

CoNeCo

Concurrency, Networks, and Coinduction

Introduction

Kleene Algebra can reason about *program flow*.

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

Introduction

Kleene Algebra can reason about *program flow*.

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

Concurrent Kleene Algebra: $\parallel$

Introduction

Kleene Algebra can reason about *program flow*.

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

Concurrent Kleene Algebra: $\parallel$

Start: $x = y = 0$

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

End: $r_1 \neq 0$ xor $r_2 \neq 0$

Introduction

Kleene Algebra can reason about *program flow*.

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

Concurrent Kleene Algebra: $\parallel$

Start: $x = y = 0$

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

End: $r_1 \neq 0$ xor $r_2 \neq 0$

CKA results

- not much . . .
- Struth and collaborators
- Understanding of foundations missing

Next: A small step towards foundations of Concurrent KA.

Introduction

Kleene theorem:

$$\begin{tikzcd}[column sep=huge] \begin{array}{c}\text{terms}\\ \text{denotation}\end{array} \arrow[r, Rightarrow, bend left] & \begin{array}{c}\text{automata}\\ \text{operation}\end{array} \arrow[l, Rightarrow, bend left] \end{tikzcd}$$

Introduction

Existing Kleene Theoremshave . . .

- . . . non-deterministic automata
- . . . *semantic* preconditions

Introduction

Existing Kleene Theoremshave . . .

- . . . non-deterministic automata
- . . . *semantic* preconditions

Our Kleene Theorem has . . .

- . . . (sequentially) deterministic automata
- . . . *syntactically* preconditions

Preliminaries

$\mathcal{T}$ 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

$\mathcal{T}$ 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

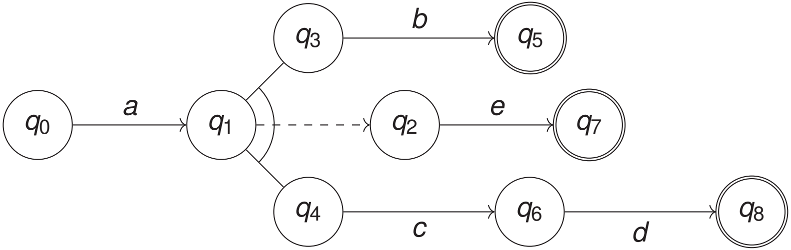
$$a \cdot (b \parallel c) \cdot d = \begin{tikzcd} & b \arrow[dr] & \\ a \arrow[ur] \arrow[dr] & & d \\ & c \arrow[ur] & \end{tikzcd}$$

Preliminaries

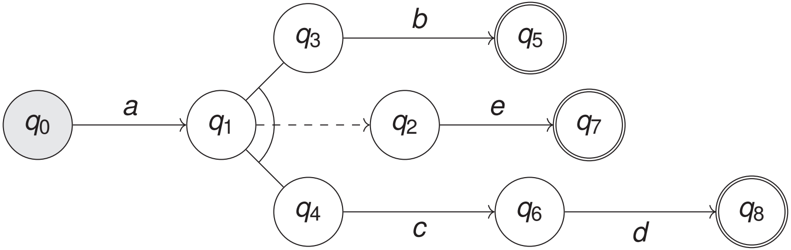
$$[\![-]\!] : \mathcal{T} \to 2^{\mathsf{Pom}_\Sigma}$$

$$\begin{aligned}
[\![0]\!] &= \emptyset \\
[\![1]\!] &= \{1\} \\
[\![a]\!] &= \{a\} \\
[\![e + f]\!] &= [\![e]\!] \cup [\![f]\!] \\
[\![e \cdot f]\!] &= [\![e]\!] \cdot [\![f]\!] \\
[\![e \parallel f]\!] &= [\![e]\!] \parallel [\![f]\!] \\
[\![e^*]\!] &= [\![e]\!]^*
\end{aligned}$$

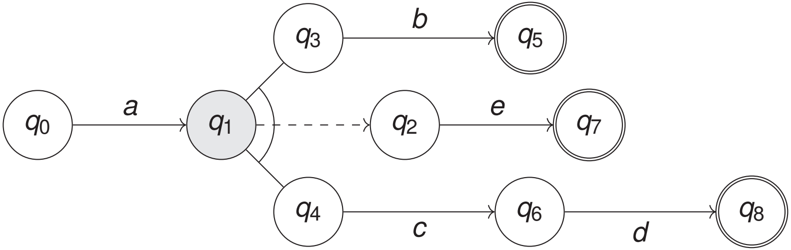
Pomset Automata



Pomset Automata

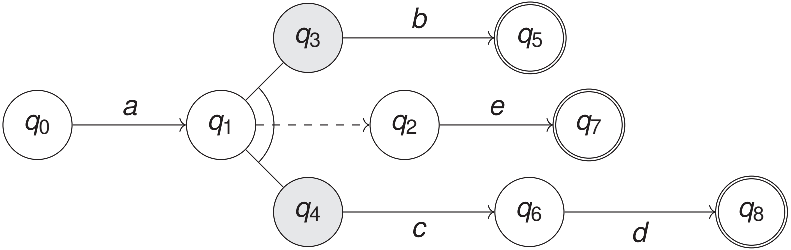


Pomset Automata



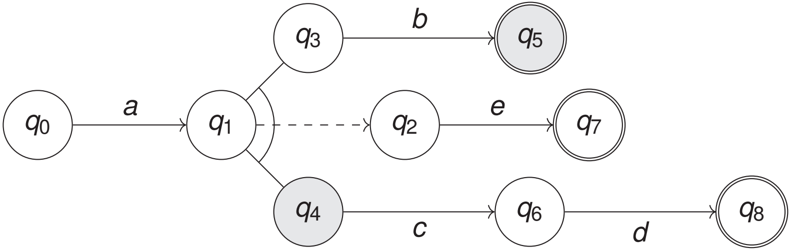
a

Pomset Automata



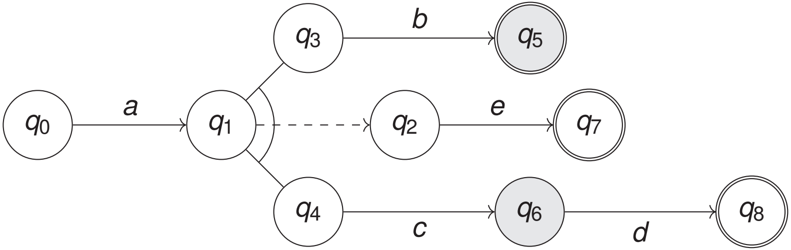
a

Pomset Automata



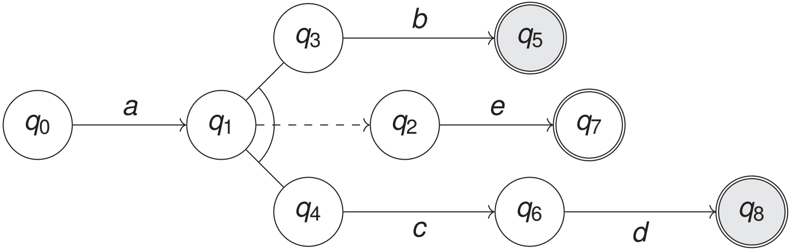
a

Pomset Automata



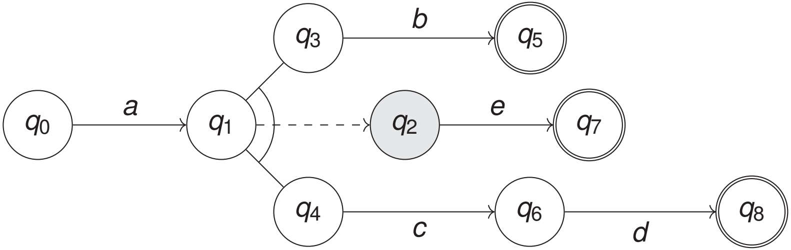
a

Pomset Automata



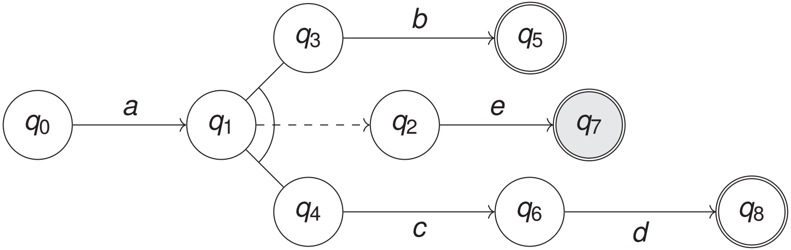
a

Pomset Automata



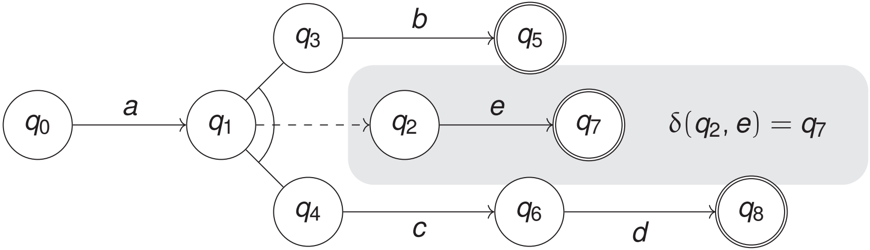
$$a(b \parallel cd)$$

Pomset Automata

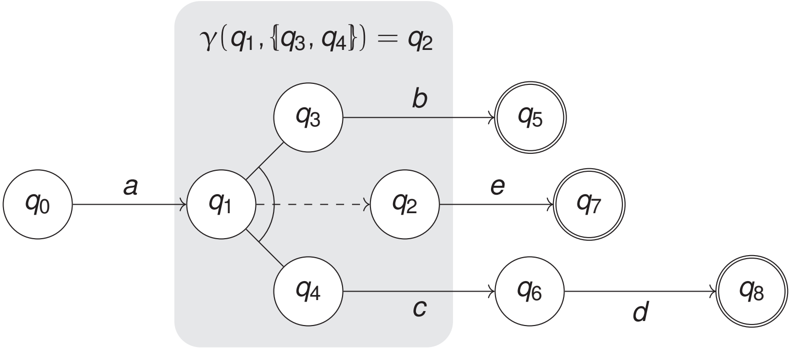


$$a(b \parallel cd)e$$

Pomset Automata



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*,
- $\delta : Q \times \Sigma \to Q$, the *sequential transition function*,
- $\gamma : Q \times \mathcal{M}_\omega(Q) \to Q$, a the *parallel transition function*

Pomset Automata

Definition

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

$$\frac{}{q \xrightarrow{a}\!\!\twoheadrightarrow_A \delta(q, a)} \qquad \frac{q \xrightarrow{U}\!\!\twoheadrightarrow_A q'' \qquad q'' \xrightarrow{V}\!\!\twoheadrightarrow_A q'}{q \xrightarrow{U \cdot V}\!\!\twoheadrightarrow_A q'} \qquad \frac{r \xrightarrow{U}\!\!\twoheadrightarrow_A r' \in F \qquad s \xrightarrow{V}\!\!\twoheadrightarrow_A s' \in F}{q \xrightarrow{U \parallel V}\!\!\twoheadrightarrow_A \gamma(q, \{\!|r, s|\!\})}$$

Pomset Automata

Definition

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

$$\frac{}{q \xrightarrow{a}\!\!\!\!\rightarrow_A \delta(q, a)} \qquad \frac{q \xrightarrow{U}\!\!\!\!\rightarrow_A q'' \qquad q'' \xrightarrow{V}\!\!\!\!\rightarrow_A q'}{q \xrightarrow{U \cdot V}\!\!\!\!\rightarrow_A q'} \qquad \frac{r \xrightarrow{U}\!\!\!\!\rightarrow_A r' \in F \qquad s \xrightarrow{V}\!\!\!\!\rightarrow_A s' \in F}{q \xrightarrow{U \parallel V}\!\!\!\!\rightarrow_A \gamma(q, \{\!|r, s|\!\})}$$

$L_A(q)$ is defined by

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

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. $\delta_\Sigma : \mathcal{T} \times \Sigma \to \mathcal{T}$: sequential derivatives

Expressions to Automata

Brzozowski-construction:

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

Expressions to Automata

Brzozowski-construction:

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

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

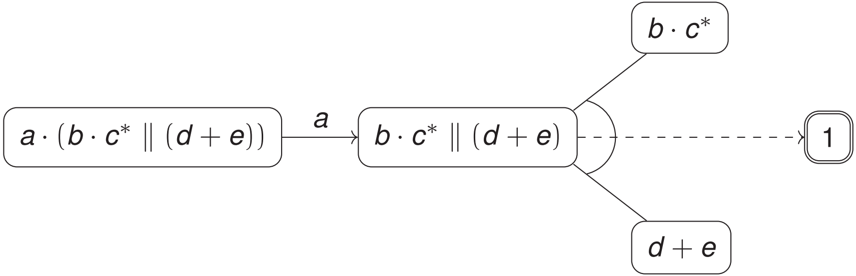
Expressions to Automata

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

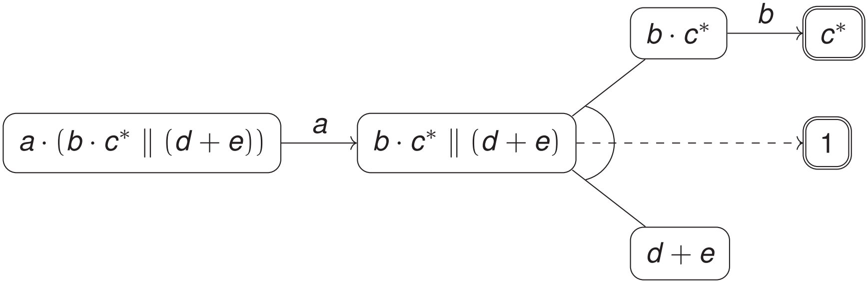
Expressions to Automata

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

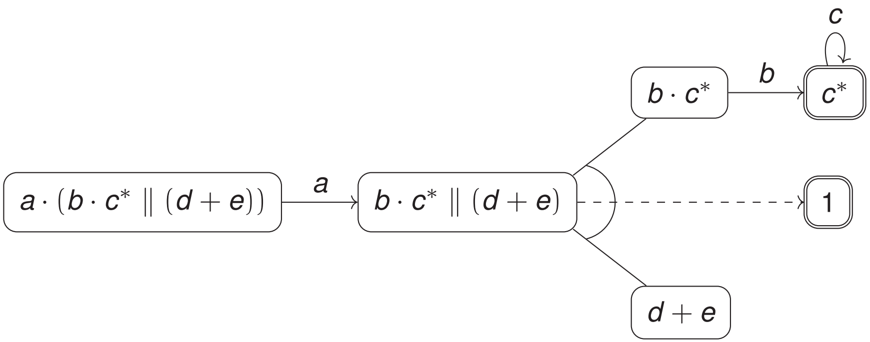
Expressions to Automata



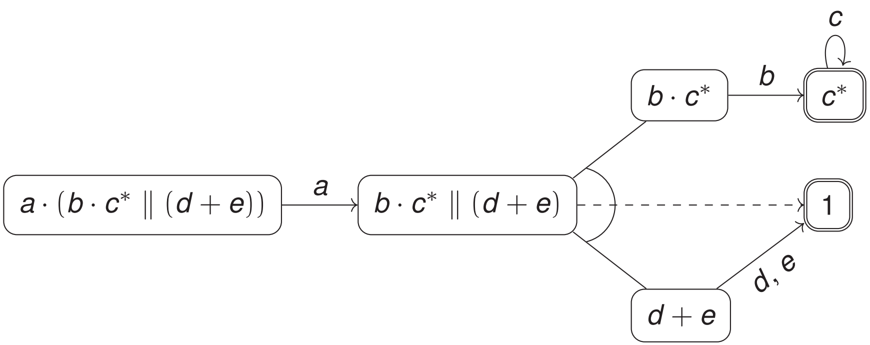
Expressions to Automata



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 $L_\Sigma(e) = [\![e]\!]$.

Proof in two parts:

Expressions to Automata

Theorem

Let $e \in \mathcal{T}$; then $L_\Sigma(e) = [\![e]\!]$.

Proof in two parts:

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

Expressions to Automata

Theorem

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

Proof in two parts:

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

Expressions to Automata

Theorem

Let $e \in \mathcal{T}$; then $L_\Sigma(e) = [\![e]\!]$.

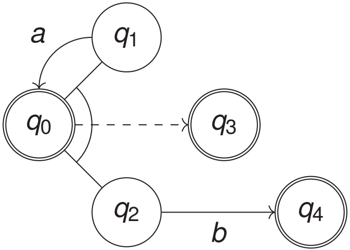
Proof in two parts:

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

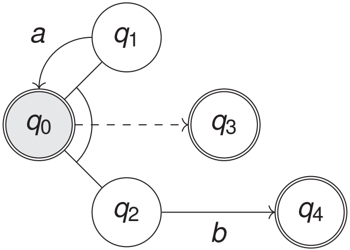
Theorem

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

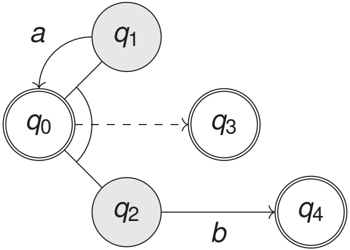
Automata to Expressions



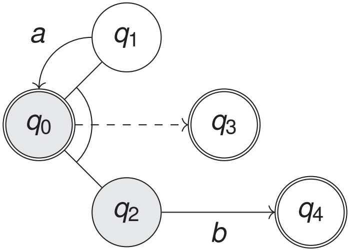
Automata to Expressions



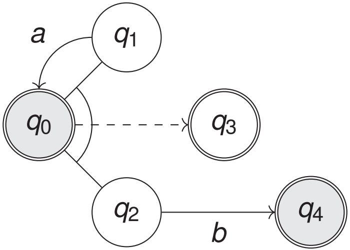
Automata to Expressions



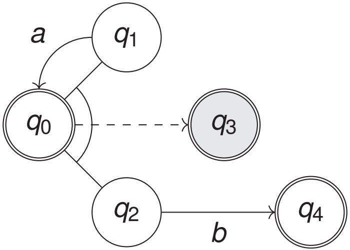
Automata to Expressions



Automata to Expressions

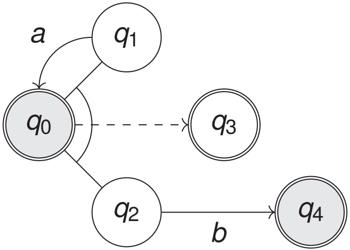


Automata to Expressions

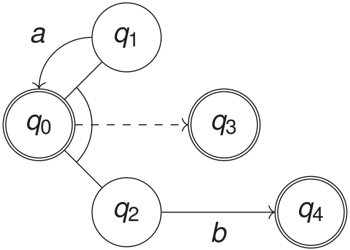


$$a \parallel b$$

Automata to Expressions

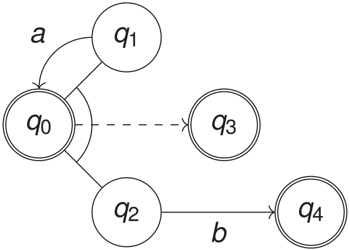


Automata to Expressions



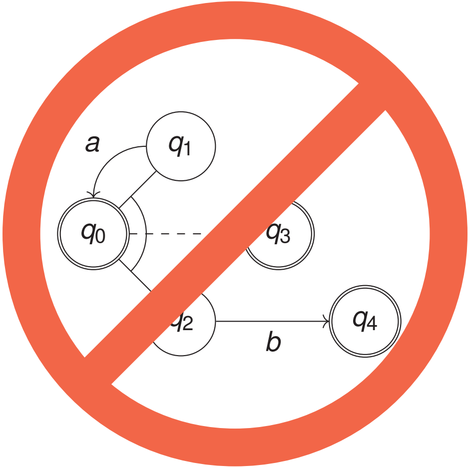
$$a(a \parallel b) \parallel b$$

Automata to Expressions

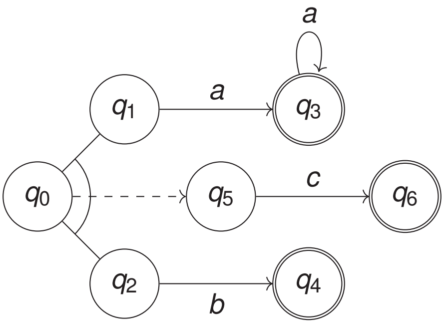


$$a(a(a \parallel b)) \parallel b$$

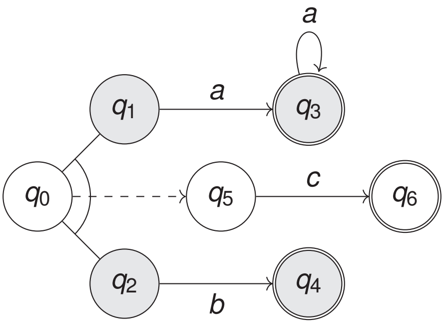
Automata to Expressions



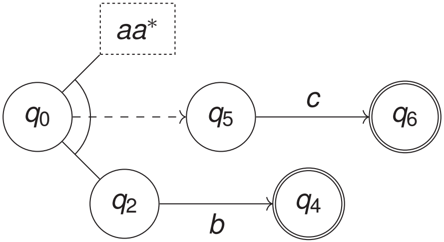
Automata to Expressions



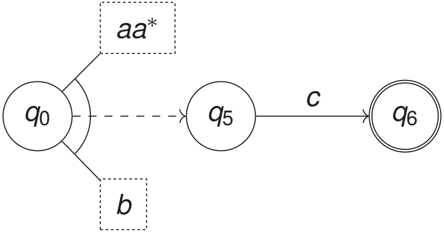
Automata to Expressions



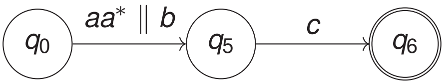
Automata to Expressions



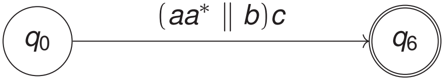
Automata to Expressions



Automata to Expressions



Automata to Expressions



Further Work

Extend to WCKA: exchange law.

Further Work

Extend to WCKA: exchange law.

Possible applications

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

GoNeCo

Further Work

Extend to WCKA: exchange law.

Possible applications

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

Endgame: concurrent extension of NetKAT.

GoNeCo