Communicating Finite-State Machines and Two-Variable Logic

Benedikt Bollig, Marie Fortin, and Paul Gastin

LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, France

STACS 2018
Caen, France
February 28 – March 3

Büchi-Elgot-Trakhtenbrot theorem ('60s)

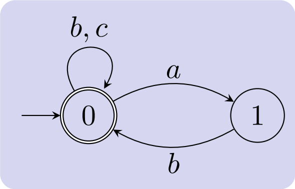
Büchi-Elgot-Trakhtenbrot theorem ('60s)

$$b \rightarrow a \rightarrow b \rightarrow c \rightarrow a \rightarrow b \rightarrow c$$

Words

$$\forall x.\ a(x) \Rightarrow \exists y.\ (x \rightarrow y \wedge b(y))$$

Monadic Second-Order Logic



Finite automaton

MSO$[\rightarrow]$ = Finite Automata

Büchi-Elgot-Trakhtenbrot theorem ('60s)

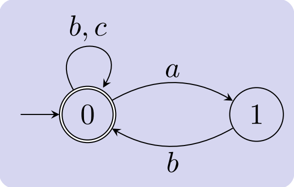
$$b \to a \to b \to c \to a \to b \to c$$

Words

$$\forall x.\ a(x) \Rightarrow \exists y.\ (x \to y \wedge b(y))$$

Monadic Second-Order Logic

induction



Finite automaton

MSO$[\to]$ = Finite Automata

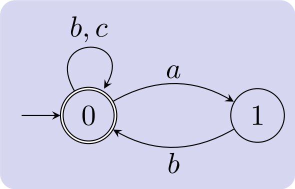
Büchi-Elgot-Trakhtenbrot theorem ('60s)

$$b \rightarrow a \rightarrow b \rightarrow c \rightarrow a \rightarrow b \rightarrow c$$

Words

$$\begin{aligned} & \exists X_0.\ \exists X_1.\ \forall x.\ \forall y. \\ & \qquad x \rightarrow y \wedge X_0(x) \wedge a(y) \Rightarrow X_1(y) \\ \wedge & \qquad x \rightarrow y \wedge X_0(x) \wedge b(y) \Rightarrow X_0(y) \\ \wedge & \qquad \ldots \end{aligned}$$

induction



Monadic Second-Order Logic

Finite automaton

MSO$[\rightarrow]$ = Finite Automata

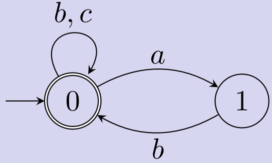
Büchi-Elgot-Trakhtenbrot theorem ('60s)

$$b \rightarrow a \rightarrow b \rightarrow c \rightarrow a \rightarrow b \rightarrow c$$

Words

$$\begin{aligned} &\exists X_0.\ \exists X_1.\ \forall x.\ \forall y. \\ &\quad x \rightarrow y \wedge X_0(x) \wedge a(y) \Rightarrow X_1(y) \\ \wedge &\quad x \rightarrow y \wedge X_0(x) \wedge b(y) \Rightarrow X_0(y) \\ \wedge &\quad \ldots \end{aligned}$$

induction



Monadic Second-Order Logic

Finite automaton

MSO$[\rightarrow]$ = Finite Automata = EMSO$[\rightarrow]$

$\exists X_0 \ldots \exists X_n.\varphi$ with $\varphi \in$ FO$[\rightarrow]$

Büchi-Elgot-Trakhtenbrot theorem ('60s)

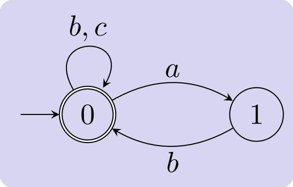
$$b \to a \to b \to c \to a \to b \to c$$

Words

$$\exists X_0.\ \exists X_1.\ \forall x.\ \forall y.$$
$$\begin{array}{ll} & x \to y \wedge X_0(x) \wedge a(y) \Rightarrow X_1(y) \\ \wedge & x \to y \wedge X_0(x) \wedge b(y) \Rightarrow X_0(y) \\ \wedge & \ldots \end{array}$$

Monadic Second-Order Logic

induction



Finite automaton

MSO[$\to$] = Finite Automata = EMSO[$\to$] = EMSO^2[$\to$]

$\exists X_0 \ldots \exists X_n.\varphi$ with $\varphi \in$ FO[$\to$]

two first-order variable names

. . . and some of its extensions

. . . and some of its extensions

Over trees:

MSO = Tree Automata = EMSO^2 [Thatcher-Wright '68]

. . . and some of its extensions

Over trees:
MSO = Tree Automata = EMSO^2 [Thatcher-Wright '68]

Over Mazurkievitch traces:
MSO = Asynchronous Automata = EMSO^2 [Thomas '90]

. . . and some of its extensions

Over trees:
MSO = Tree Automata = EMSO^2 [Thatcher-Wright '68]

Over Mazurkievitch traces:
MSO = Asynchronous Automata = EMSO^2 [Thomas '90]

Over data words:
EMSO^2 = Data Automata $\subsetneq$ EMSO
[Bojanczyk-David-Muscholl-Schwentick-Segoufin '06]

. . . and some of its extensions

Over trees:
$\mathrm{MSO} = \text{Tree Automata} = \mathrm{EMSO}^2$ [Thatcher-Wright '68]

Over Mazurkievitch traces:
$\mathrm{MSO} = \text{Asynchronous Automata} = \mathrm{EMSO}^2$ [Thomas '90]

Over data words:
$\mathrm{EMSO}^2 = \text{Data Automata} \subsetneq \mathrm{EMSO}$
[Bojanczyk-David-Muscholl-Schwentick-Segoufin '06]

. . .

Goal: A Büchi-like theorem for message-passing systems

Communicating finite-state machines (CFMs)

[Brand–Zafiropulo '83]

Communicating finite-state machines (CFMs)

[Brand–Zafiropulo '83]

- Finite set of processes, e.g. $Proc = \{p, q, r\}$

Communicating finite-state machines (CFMs)

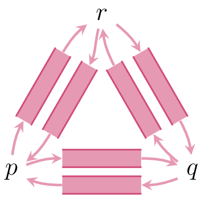
[Brand–Zafiropulo '83]

- Finite set of processes, e.g. $Proc = \{p, q, r\}$
- Unbounded point-to-point FIFO channels

Communicating finite-state machines (CFMs)

[Brand–Zafiropulo '83]

- Finite set of processes, e.g. $Proc = \{p, q, r\}$
- Unbounded point-to-point FIFO channels

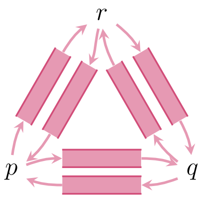


Communicating finite-state machines (CFMs)

[Brand–Zafiropulo '83]

- Finite set of processes, e.g. $Proc = \{p, q, r\}$
- Unbounded point-to-point FIFO channels
- For each process, one finite labeled transition system over

$$(\underbrace{\Sigma}_{\text{Finite alphabet}} \times \{\underbrace{!}_{\text{send}}, \underbrace{?}_{\text{receive}}\} \times \underbrace{Msg}_{\text{Finite set of messages}} \times \underbrace{Proc}_{\text{Processes}}) \cup \Sigma$$



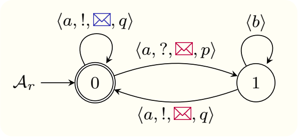
Communicating finite-state machines (CFMs)

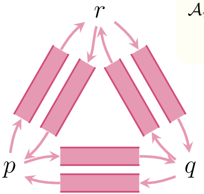
[Brand–Zafiropulo '83]

- Finite set of processes, e.g. $Proc = \{p, q, r\}$
- Unbounded point-to-point FIFO channels
- For each process, one finite labeled transition system over

$$(\Sigma \times \{!, ?\} \times Msg \times Proc) \cup \Sigma$$

Finite alphabet — send — receive — Finite set of messages — Processes





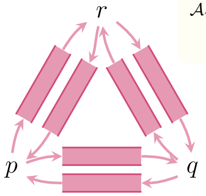
Communicating finite-state machines (CFMs)

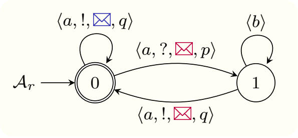
[Brand–Zafiropulo '83]

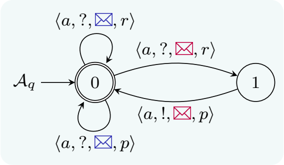
- Finite set of processes, e.g. $Proc = \{p, q, r\}$
- Unbounded point-to-point FIFO channels
- For each process, one finite labeled transition system over

$$(\Sigma \times \{!, ?\} \times Msg \times Proc) \cup \Sigma$$

Finite alphabet — send — receive — Finite set of messages — Processes





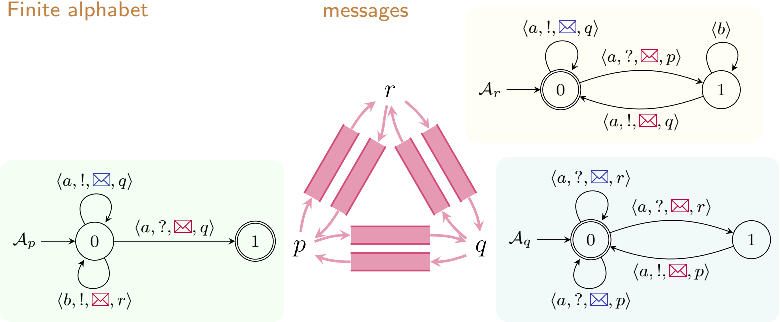


Communicating finite-state machines (CFMs)

[Brand–Zafiropulo '83]

- Finite set of processes, e.g. $Proc = \{p, q, r\}$
- Unbounded point-to-point FIFO channels
- For each process, one finite labeled transition system over

$$(\underbrace{\Sigma}_{\text{Finite alphabet}} \times \{\underbrace{!}_{\text{send}}, \underbrace{?}_{\text{receive}}\} \times \underbrace{Msg}_{\text{Finite set of messages}} \times \underbrace{Proc}_{\text{Processes}}) \cup \Sigma$$



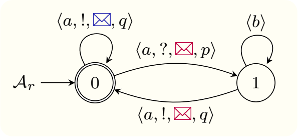
Communicating finite-state machines (CFMs)

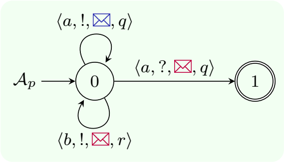
[Brand–Zafiropulo '83]

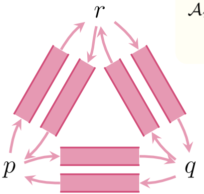
- Finite set of processes, e.g. $Proc = \{p, q, r\}$
- Unbounded point-to-point FIFO channels
- For each process, one finite labeled transition system over

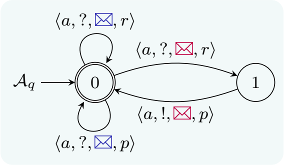
$$(\underbrace{\Sigma}_{\text{Finite alphabet}} \times \{\underbrace{!}_{\text{send}}, \underbrace{?}_{\text{receive}}\} \times \underbrace{Msg}_{\text{Finite set of messages}} \times \underbrace{Proc}_{\text{Processes}}) \cup \Sigma$$

- Global acceptance condition



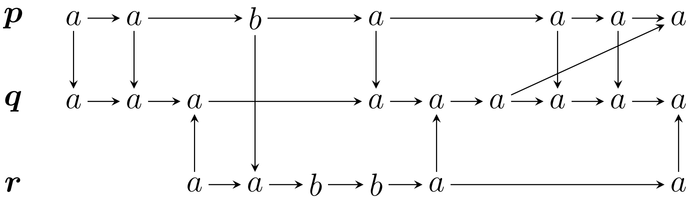






Language of a CFM

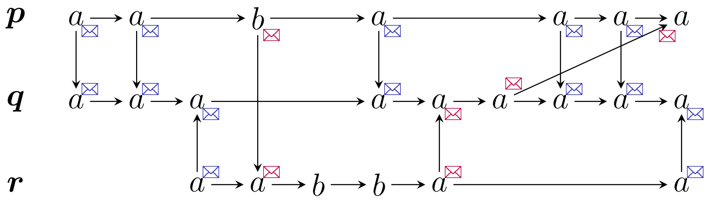
CFMs recognize languages of **Message Sequence Charts**:



- FIFO
- acyclic

Language of a CFM

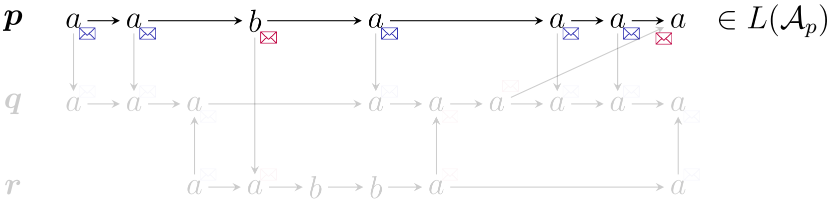
CFMs recognize languages of **Message Sequence Charts**:



- FIFO
- acyclic

Language of a CFM

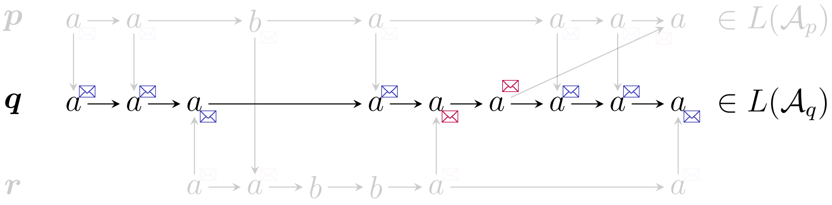
CFMs recognize languages of **Message Sequence Charts**:



- FIFO
- acyclic

Language of a CFM

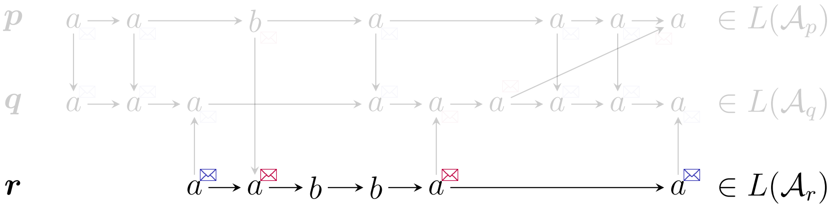
CFMs recognize languages of **Message Sequence Charts**:



- FIFO
- acyclic

Language of a CFM

CFMs recognize languages of **Message Sequence Charts**:



- FIFO
- acyclic

Monadic Second-Order Logic over MSCs

Monadic Second-Order Logic over MSCs

MSO$[\rightarrow, \rightarrow^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

Monadic Second-Order Logic over MSCs

MSO$[\rightarrow, \rightarrow^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

- FO$[\rightarrow, \rightarrow^*]$: no $\exists X$.

Monadic Second-Order Logic over MSCs

MSO$[\to, \to^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \to y \mid x \to^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

- FO$[\to, \to^*]$: no $\exists X$.
- FO$[\to]$: no $x \to^* y$

Monadic Second-Order Logic over MSCs

MSO$[\rightarrow, \rightarrow^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

- FO$[\rightarrow, \rightarrow^*]$: no $\exists X.$
- FO$[\rightarrow]$: no $x \rightarrow^* y$
- FO$^2[\rightarrow, \rightarrow^*]$: only two variable names

Monadic Second-Order Logic over MSCs

MSO$[\rightarrow, \rightarrow^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

- FO$[\rightarrow, \rightarrow^*]$: no $\exists X$.
- FO$[\rightarrow]$: no $x \rightarrow^* y$
- FO$^2[\rightarrow, \rightarrow^*]$: only two variable names
- EMSO$[\rightarrow, \rightarrow^*]$: $\exists X_1 \ldots \exists X_n.\ \varphi$, with $\varphi \in$ FO$[\rightarrow, \rightarrow^*]$

Monadic Second-Order Logic over MSCs

MSO$[\rightarrow, \rightarrow^*]$

$$\begin{array}{rcl} \varphi & ::= & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi \end{array}$$

- FO$[\rightarrow, \rightarrow^*]$: no $\exists X$.
- FO$[\rightarrow]$: no $x \rightarrow^* y$
- FO$^2[\rightarrow, \rightarrow^*]$: only two variable names
- EMSO$[\rightarrow, \rightarrow^*]$: $\exists X_1 \ldots \exists X_n.\ \varphi$, with $\varphi \in$ FO$[\rightarrow, \rightarrow^*]$
- EMSO$^2[\rightarrow, \rightarrow^*]$: $\exists X_1 \ldots \exists X_n.\ \varphi$, with $\varphi \in$ FO$^2[\rightarrow, \rightarrow^*]$

Monadic Second-Order Logic over MSCs

$\mathsf{MSO}[\rightarrow, \rightarrow^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

- $\mathsf{FO}[\rightarrow, \rightarrow^*]$: no $\exists X$.
- $\mathsf{FO}[\rightarrow]$: no $x \rightarrow^* y$
- $\mathsf{FO}^2[\rightarrow, \rightarrow^*]$: only two variable names
- $\mathsf{EMSO}[\rightarrow, \rightarrow^*]$: $\exists X_1 \ldots \exists X_n.\ \varphi$, with $\varphi \in \mathsf{FO}[\rightarrow, \rightarrow^*]$
- $\mathsf{EMSO}^2[\rightarrow, \rightarrow^*]$: $\exists X_1 \ldots \exists X_n.\ \varphi$, with $\varphi \in \mathsf{FO}^2[\rightarrow, \rightarrow^*]$
- ...

Monadic Second-Order Logic over MSCs

MSO$[\rightarrow, \rightarrow^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

Examples in FO$^2[\rightarrow, \rightarrow^*]$:

Monadic Second-Order Logic over MSCs

$\mathsf{MSO}[\rightarrow, \rightarrow^*]$

$$\begin{aligned} \varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi \end{aligned}$$

Examples in $\mathsf{FO}^2[\rightarrow, \rightarrow^*]$:

- $x \parallel y \ \equiv\ \neg(x \rightarrow^* y) \wedge \neg(y \rightarrow^* x)$

Monadic Second-Order Logic over MSCs

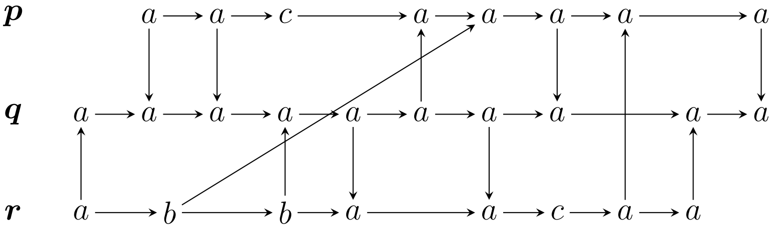
MSO$[\rightarrow, \rightarrow^*]$

$$\begin{aligned}\varphi \quad ::= \quad & a(x) \mid p(x) \mid x = y \mid x \rightarrow y \mid x \rightarrow^* y \\ & \mid \exists x.\ \varphi \mid \exists X.\ \varphi \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi\end{aligned}$$

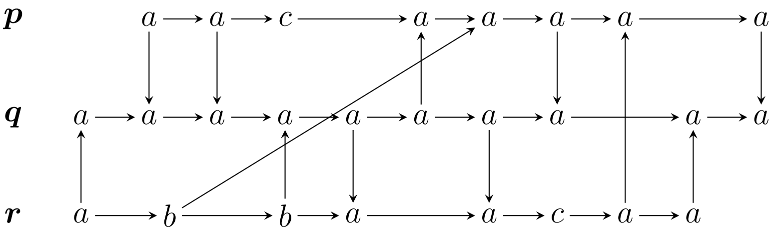
Examples in FO$^2[\rightarrow, \rightarrow^*]$:

- $x \parallel y \ \equiv\ \neg(x \rightarrow^* y) \wedge \neg(y \rightarrow^* x)$
- Mutual exclusion: $\neg(\exists x.\ \exists y.\ c(x) \wedge c(y) \wedge x \parallel y)$

Monadic Second-Order Logic over MSCs

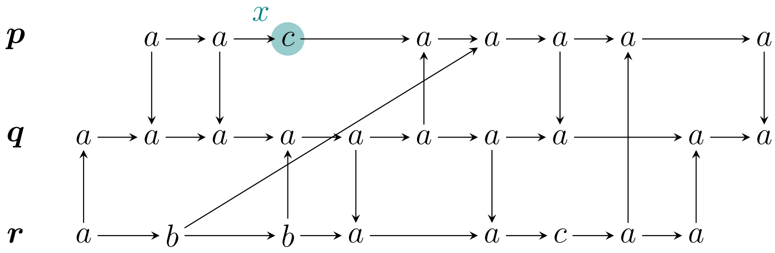


Monadic Second-Order Logic over MSCs



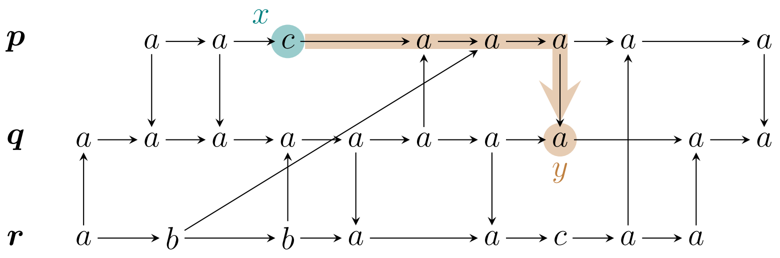
$$M \models \forall x.\ c(x) \implies \exists y.\ q(y) \wedge x \rightarrow^* y$$

Monadic Second-Order Logic over MSCs



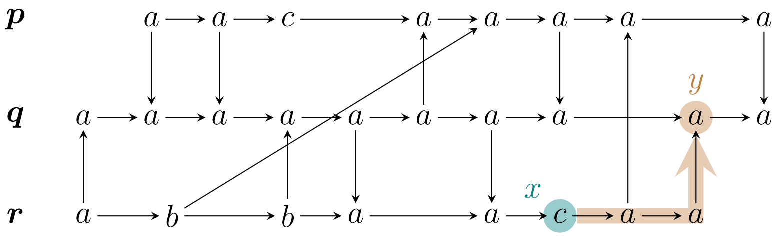
$$M \models \forall x.\ c(x) \implies \exists y.\ q(y) \wedge x \rightarrow^* y$$

Monadic Second-Order Logic over MSCs



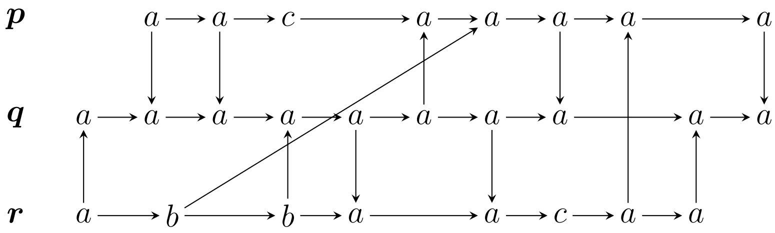
$$M \models \forall x.\ c(x) \implies \exists y.\ q(y) \wedge x \rightarrow^* y$$

Monadic Second-Order Logic over MSCs



$$M \models \forall x.\ c(x) \implies \exists y.\ q(y) \wedge x \to^* y$$

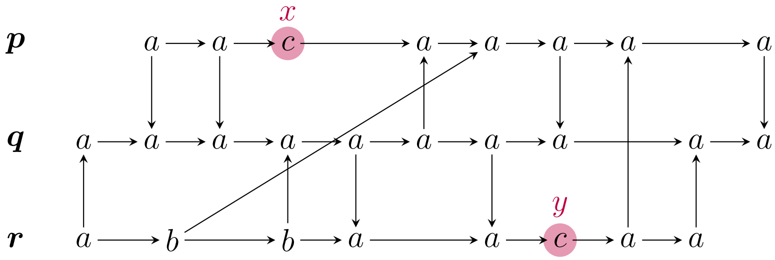
Monadic Second-Order Logic over MSCs



$$M \models \forall x.\ c(x) \implies \exists y.\ q(y) \land x \rightarrow^* y$$

$$M \not\models \neg(\exists x.\ \exists y.\ c(x) \land c(y) \land x \parallel y)$$

Monadic Second-Order Logic over MSCs



$$M \models \forall x.\ c(x) \implies \exists y.\ q(y) \wedge x \rightarrow^* y$$

$$M \not\models \neg(\exists x.\ \exists y.\ c(x) \wedge c(y) \wedge x \parallel y)$$

"Büchi Theorems" for CFMs

"Büchi Theorems" for CFMs

Theorem (Mukund et al. '05, Genest-Kuske-Muscholl '06)

When channels are **bounded**, CFM = $\mathrm{MSO}[\rightarrow]$.

"Büchi Theorems" for CFMs

Theorem (Mukund et al. '05, Genest-Kuske-Muscholl '06)

When channels are **bounded**, CFM = MSO$[\rightarrow]$.

Theorem (Bollig-Leucker '06)

CFM = EMSO$[\rightarrow] \subsetneq$ MSO$[\rightarrow]$.

"Büchi Theorems" for CFMs

Theorem (Mukund et al. '05, Genest-Kuske-Muscholl '06)

When channels are **bounded**, CFM $=$ MSO$[\to]$.

Theorem (Bollig-Leucker '06)

CFM $=$ EMSO$[\to] \subsetneq$ MSO$[\to]$.

In the unbounded case, CFMs are not complementable!
$\to$ No inductive translation logic $\rightsquigarrow$ CFMs.

"Büchi Theorems" for CFMs

Theorem (Mukund et al. '05, Genest-Kuske-Muscholl '06)

When channels are **bounded**, $\mathrm{CFM} = \mathrm{MSO}[\rightarrow]$.

Theorem (Bollig-Leucker '06)

$\mathrm{CFM} = \mathrm{EMSO}[\rightarrow] \subsetneq \mathrm{MSO}[\rightarrow]$.

In the unbounded case, CFMs are not complementable!
$\rightarrow$ No inductive translation logic $\rightsquigarrow$ CFMs.

Theorem

$\mathrm{CFM} = \mathrm{EMSO}^2[\rightarrow, \rightarrow^*]$.

Scott normal form ('65)

Scott normal form:
Any EMSO$^2[\rightarrow, \rightarrow^*]$ formula is equivalent to a formula

$$\exists X_1 . \dots . \exists X_m . \ \forall x . \forall y . \psi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x . \exists y . \psi_i(x,y)$$

where $\psi(x,y)$ and each $\psi_i(x,y)$ is quantifier-free.

From Scott normal form to CFMs

$$\exists X_1 \ldots . \exists X_m . \ \forall x . \forall y . \psi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x . \exists y . \psi_i(x,y)$$

From Scott normal form to CFMs

$$\exists X_1 \ldots . \exists X_m .\, \forall x. \forall y. \psi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x. \exists y. \psi_i(x,y)$$

- Guess a valuation for each X_i.

From Scott normal form to CFMs

$$\exists X_1 . \ldots . \exists X_m . \; \forall x . \forall y . \psi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x . \exists y . \psi_i(x,y)$$

- Guess a valuation for each X_i.
- Compute the **type** of each event.

From Scott normal form to CFMs

$$\exists X_1. \dots \exists X_m.\ \forall x.\forall y.\psi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x.\exists y.\psi_i(x,y)$$

- Guess a valuation for each X_i.
- Compute the **type** of each event.

The type of x characterizes the set of quantifier-free formulas φ with one free variable such that $\exists y.\varphi(x)$ holds, or such that $\forall y.\varphi(x)$ holds.

From Scott normal form to CFMs

$$\exists X_1 \ldots . \exists X_m .\ \forall x . \forall y . \psi(x,y) \wedge \bigwedge_{i=1}^{n} \forall x . \exists y . \psi_i(x,y)$$

- Guess a valuation for each X_i.
- Compute the **type** of each event.

The type of x characterizes the set of quantifier-free formulas φ with one free variable such that $\exists y . \varphi(x)$ holds, or such that $\forall y . \varphi(x)$ holds.

- Restrict to types verifying $\forall y . \psi(x,y)$ and all $\exists y . \psi_i(x,y)$.

From Scott normal form to CFMs

$$\exists X_1 \ldots . \exists X_m . \forall x . \forall y . \psi(x, y) \wedge \bigwedge_{i=1}^{n} \forall x . \exists y . \psi_i(x, y)$$

- Guess a valuation for each X_i.
- Compute the **type** of each event.

The type of x characterizes the set of quantifier-free formulas φ with one free variable such that $\exists y . \varphi(x)$ holds, or such that $\forall y . \varphi(x)$ holds.

- Restrict to types verifying $\forall y . \psi(x, y)$ and all $\exists y . \psi_i(x, y)$.

Type of an event

Type of event x: set of all possible tuples $(\bowtie, p, a)$ such that for some event y:

- $x \bowtie y$
- x is on process p, and labeled a.

$\bowtie$ is one of all possible relative positions between two events:

Type of an event

Type of event x: set of all possible tuples $(\bowtie, p, a)$ such that for some event y:

- $x \bowtie y$
- x is on process p, and labeled a.

$\bowtie$ is one of all possible relative positions between two events:

- $x = y$

Type of an event

Type of event x: set of all possible tuples $(\bowtie, p, a)$ such that for some event y:

- $x \bowtie y$
- x is on process p, and labeled a.

$\bowtie$ is one of all possible relative positions between two events:

- $x = y$
- $x \rightarrow y$

Type of an event

Type of event x: set of all possible tuples $(\bowtie, p, a)$ such that for some event y:

- $x \bowtie y$
- x is on process p, and labeled a.

$\bowtie$ is one of all possible relative positions between two events:

- $x = y$
- $x \rightarrow y$
- $y \rightarrow x$

Type of an event

Type of event x: set of all possible tuples $(\bowtie, p, a)$ such that for some event y:

- $x \bowtie y$
- x is on process p, and labeled a.

$\bowtie$ is one of all possible relative positions between two events:

- $x = y$
- $x \rightarrow y$
- $y \rightarrow x$
- $x \rightsquigarrow y$, where $\rightsquigarrow \, = \, \rightarrow^+ \setminus \rightarrow$

Type of an event

Type of event x: set of all possible tuples $(\bowtie, p, a)$ such that for some event y:

- $x \bowtie y$
- x is on process p, and labeled a.

$\bowtie$ is one of all possible relative positions between two events:

- $x = y$
- $x \rightarrow y$
- $y \rightarrow x$
- $x \rightsquigarrow y$, where $\rightsquigarrow \, = \, \rightarrow^+ \setminus \rightarrow$
- $y \rightsquigarrow x$

Type of an event

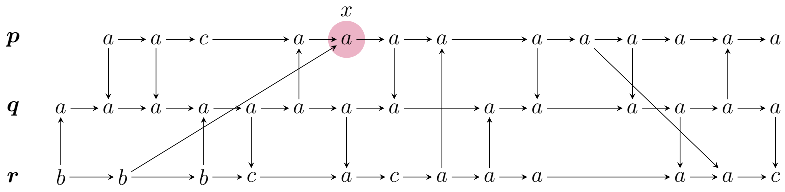
Type of event x: set of all possible tuples $(\bowtie, p, a)$ such that for some event y:

- $x \bowtie y$
- x is on process p, and labeled a.

$\bowtie$ is one of all possible relative positions between two events:

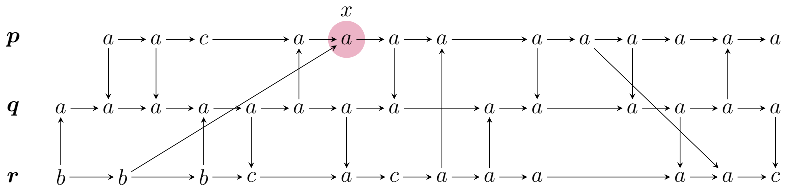
- $x = y$
- $x \to y$
- $y \to x$
- $x \rightsquigarrow y$, where $\rightsquigarrow \; = \; \to^+ \setminus \to$
- $y \rightsquigarrow x$
- $x \parallel y$, i.e., $x \not\to^* y$ and $y \not\to^* x$

Type of an event



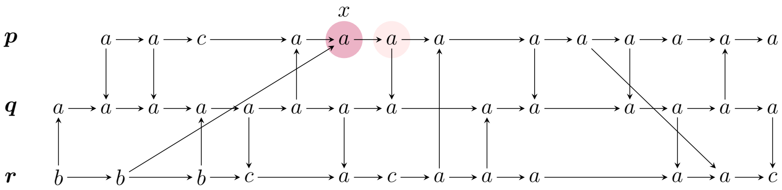
Type of an event

$x = y \ : \ (p, a)$



Type of an event

$x = y \ : \ (p, a)$ $\qquad$ $x \rightarrow y \ : \ (p, a)$

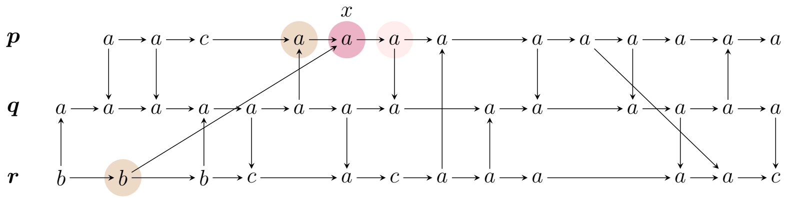


Type of an event

$x = y \;:\; (p, a)$

$x \rightarrow y \;:\; (p, a)$

$y \rightarrow x \;:\; (p, a), (r, b)$



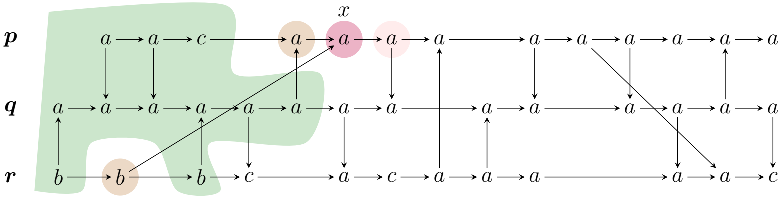
Type of an event

$x = y \;:\; (p, a)$

$x \rightarrow y \;:\; (p, a)$

$y \rightarrow x \;:\; (p, a), (r, b)$

$y \rightsquigarrow x \;:\; (p, a), (p, c), (q, a), (r, b)$



Type of an event

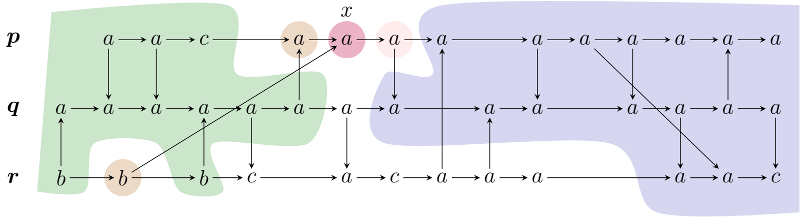
$x = y \;:\; (p, a)$

$x \rightarrow y \;:\; (p, a)$

$y \rightarrow x \;:\; (p, a), (r, b)$

$y \rightsquigarrow x \;:\; (p, a), (p, c), (q, a), (r, b)$

$x \rightsquigarrow y \;:\; (p, a), (q, a), (r, a), (r, c)$



Type of an event

$x = y \;:\; (p,a)$

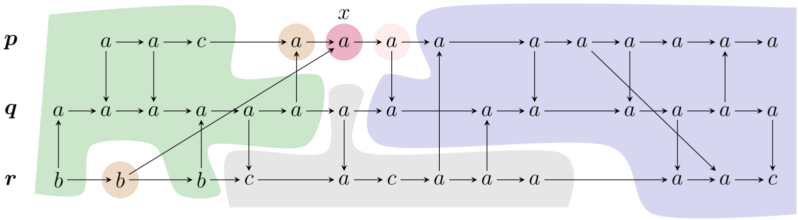
$x \rightarrow y \;:\; (p,a)$

$y \rightarrow x \;:\; (p,a),(r,b)$

$y \rightsquigarrow x \;:\; (p,a),(p,c),(q,a),(r,b)$

$x \rightsquigarrow y \;:\; (p,a),(q,a),(r,a),(r,c)$

$x \parallel y \;:\; (q,a),(r,a),(r,c)$



Type of an event

$x = y \ : \ (p,a)$

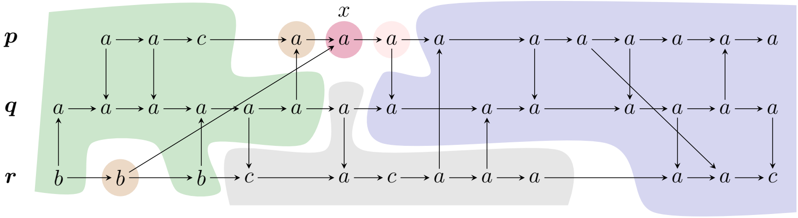
$x \rightarrow y \ : \ (p,a)$

$y \rightarrow x \ : \ (p,a),(r,b)$

$y \rightsquigarrow x \ : \ (p,a),(p,c),(q,a),(r,b)$

$x \rightsquigarrow y \ : \ (p,a),(q,a),(r,a),(r,c)$

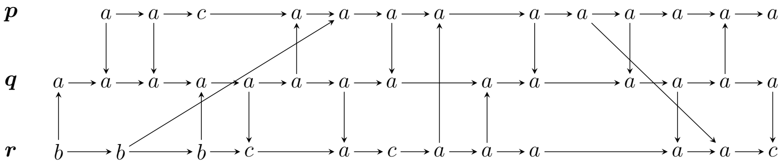
$x \parallel y \ : \ (q,a),(r,a),(r,c)$



$$\models \exists y.\ \neg(x \rightarrow^* y) \wedge \neg p(y) \wedge c(y)$$

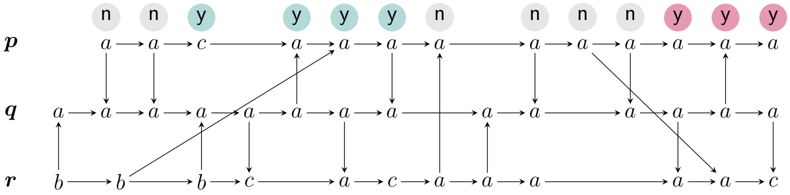
Labels of parallel events

Automaton that determines the set of events on process p that are parallel to some c on process r:



Labels of parallel events

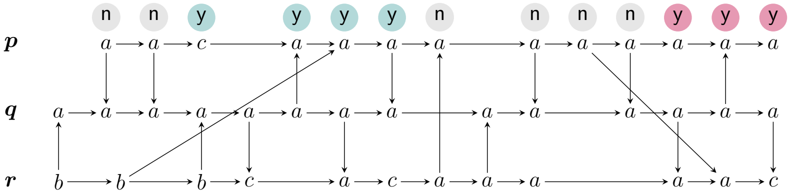
Automaton that determines the set of events on process p that are parallel to some c on process r:



Guess yes/no for each event

Labels of parallel events

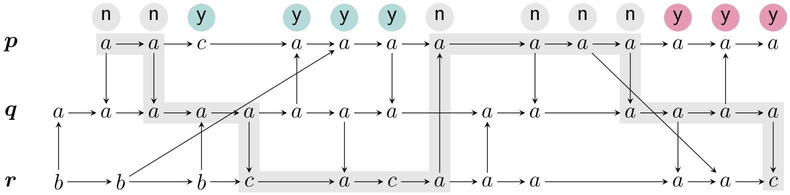
Automaton that determines the set of events on process p that are parallel to some c on process r:



One component checks that “no” guesses are correct

Labels of parallel events

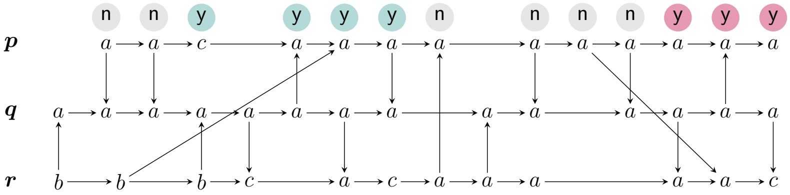
Automaton that determines the set of events on process p that are parallel to some c on process r:



One component checks that "no" guesses are correct
$\to$ path containing all "no"'s on p and all c's on r

Labels of parallel events

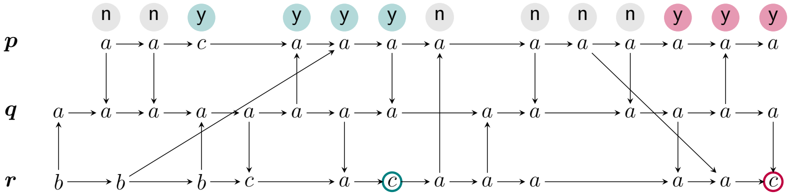
Automaton that determines the set of events on process p that are parallel to some c on process r:



One component checks that "yes" guesses are correct:

Labels of parallel events

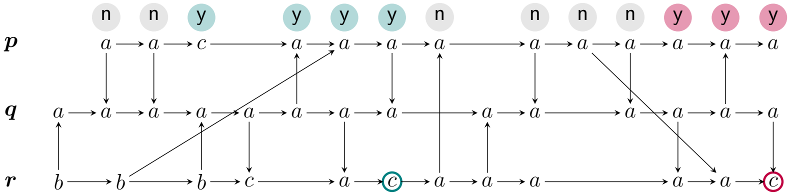
Automaton that determines the set of events on process p that are parallel to some c on process r:



One component checks that “yes” guesses are correct:

Labels of parallel events

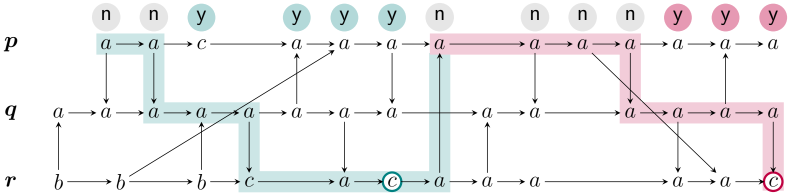
Automaton that determines the set of events on process p that are parallel to some c on process r:



One component checks that “yes” guesses are correct:
$\rightarrow$ check y $\parallel$ c, and y $\parallel$ c

Labels of parallel events

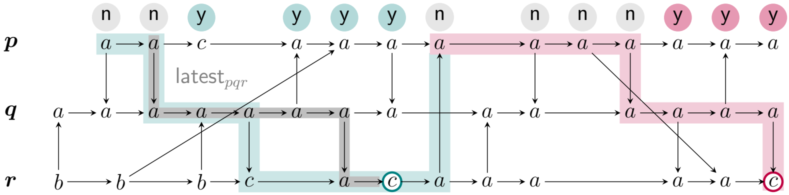
Automaton that determines the set of events on process p that are parallel to some c on process r:



One component checks that “yes” guesses are correct:
$\rightarrow$ check y $\|$ c, and y $\|$ c

Labels of parallel events

Automaton that determines the set of events on process p that are parallel to some c on process r:



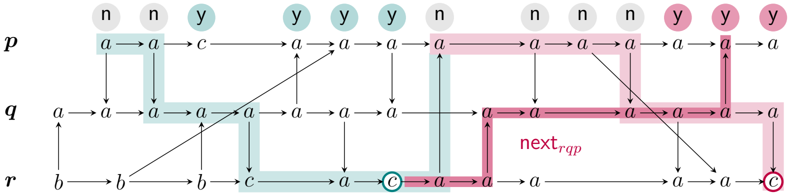
One component checks that “yes” guesses are correct:

$\to$ check y $\|$ c, and y $\|$ c

$\to$ check color of latest events co-reachable via pqr, resp. pr, and first events reachable via rqp, resp. rp

Labels of parallel events

Automaton that determines the set of events on process p that are parallel to some c on process r:



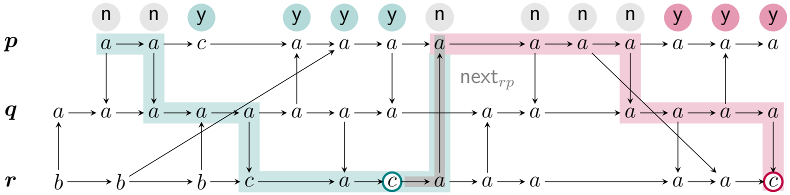
One component checks that "yes" guesses are correct:

$\rightarrow$ check y $\parallel$ c, and y $\parallel$ c

$\rightarrow$ check color of latest events co-reachable via pqr, resp. pr, and first events reachable via rqp, resp. rp

Labels of parallel events

Automaton that determines the set of events on process p that are parallel to some c on process r:



One component checks that "yes" guesses are correct:

$\rightarrow$ check y $\parallel$ c, and y $\parallel$ c

$\rightarrow$ check color of latest events co-reachable via pqr, resp. pr, and first events reachable via rqp, resp. rp

Conclusion

Conclusion

- Generalization of Büchi-Elgot-Trakhtenbrot Theorem: $\mathsf{EMSO}^2[\rightarrow, \rightarrow^*] = \mathsf{CFM}$.

Conclusion

- Generalization of Büchi-Elgot-Trakhtenbrot Theorem: $\mathsf{EMSO}^2[\rightarrow, \rightarrow^*] = \mathsf{CFM}$.
- Doubly-exponential w.r.t. the input formula (optimal).

Conclusion

- Generalization of Büchi-Elgot-Trakhtenbrot Theorem: EMSO$^2[\rightarrow, \rightarrow^*] =$ CFM.
- Doubly-exponential w.r.t. the input formula (optimal).

Open questions:

Conclusion

- Generalization of Büchi-Elgot-Trakhtenbrot Theorem: $\mathsf{EMSO}^2[\rightarrow, \rightarrow^*] = \mathsf{CFM}$.
- Doubly-exponential w.r.t. the input formula (optimal).

Open questions:

- $\mathsf{FO}[\rightarrow, \rightarrow^*] \subseteq \mathsf{CFM}$?
  $\mathsf{FO}[\rightarrow^*] \subseteq \mathsf{CFM}$?

Conclusion

- Generalization of Büchi-Elgot-Trakhtenbrot Theorem: $\mathrm{EMSO}^2[\rightarrow, \rightarrow^*] = \mathrm{CFM}$.
- Doubly-exponential w.r.t. the input formula (optimal).

Open questions:

- $\mathrm{FO}[\rightarrow, \rightarrow^*] \subseteq \mathrm{CFM}$?
  $\mathrm{FO}[\rightarrow^*] \subseteq \mathrm{CFM}$?
- What are the classes of graphs on which $\mathrm{EMSO}^2[\rightarrow, \rightarrow^*]$ and graph acceptors are expressively equivalent?

Conclusion

- Generalization of Büchi-Elgot-Trakhtenbrot Theorem: $\mathrm{EMSO}^2[\to, \to^*] = \mathrm{CFM}$.
- Doubly-exponential w.r.t. the input formula (optimal).

Open questions:

- $\mathrm{FO}[\to, \to^*] \subseteq \mathrm{CFM}$?
  $\mathrm{FO}[\to^*] \subseteq \mathrm{CFM}$?
- What are the classes of graphs on which $\mathrm{EMSO}^2[\to, \to^*]$ and graph acceptors are expressively equivalent?

Thank you!