Deterministic Automata and Extensions of Weak MSO

Mikołaj Bojańczyk
Szymon Toruńczyk

University of Warsaw

Languages of infinite words

Languages of infinite words

$abaabbaaaaaba\ldots \in (a+b)^\omega$

Languages of infinite words

$abaabbaaaaaba\ldots \in (a+b)^\omega$

Language: infinitely *a*'s on odd positions

Languages of infinite words

$$\underline{a}b\underline{a}a\underline{b}b\underline{a}a\underline{a}a\underline{a}b\underline{a}\ldots \in (a+b)^{\omega}$$

exists a set of positions X

$\exists X$

Language: infinitely *a*'s on odd positions

Languages of infinite words

$\underline{a}b\underline{a}a\underline{b}b\underline{a}a\underline{a}a\underline{a}b\underline{a}\ldots \in (a+b)^{\omega}$

exists a set of positions X

contains the first position

$$\exists X \left\{ \begin{array}{l} \forall x\ \exists y{\leq}x \quad y{\in}X \\ \\ \end{array} \right.$$

Language: infinitely *a*'s on odd positions

Languages of infinite words

$\underline{a}b\underline{a}a\underline{b}b\underline{a}a\underline{a}a\underline{a}b\underline{a}\ldots \in (a+b)^\omega$

exists a set of positions X

contains the first position

$$\exists X \begin{cases} \forall x\ \exists y{\le}x \quad y{\in}X \\ \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x{\in}X \Leftrightarrow y{\notin}X) \end{cases}$$

contains every second position

Language: infinitely *a*'s on odd positions

Languages of infinite words

$$\underline{a}b\underline{a}a\underline{b}b\underline{a}a\underline{a}a\underline{a}b\underline{a}\ldots \in (a+b)^{\omega}$$

exists a set of positions X

contains the first position

$$\exists X \begin{cases} \forall x\ \exists y{\leq}x \quad y{\in}X \\ \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x{\in}X \Leftrightarrow y{\notin}X) \\ \forall x\ \exists y{\geq}x \quad a(y) \wedge y{\in}X \end{cases}$$

contains every second position

contains infinitely many *a*'s

Language: infinitely *a*'s on odd positions

Languages of infinite words

$$\underline{a}b\underline{a}a\underline{b}b\underline{a}a\underline{a}a\underline{a}b\underline{a}\ldots \in (a+b)^{\omega}$$

exists a set of positions X

contains the first position

$$\exists X \begin{cases} \forall x\ \exists y{\leq}x \quad y{\in}X \\ \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x{\in}X \Leftrightarrow y{\notin}X) \\ \forall x\ \exists y{\geq}x \quad a(y) \wedge y{\in}X \end{cases}$$

contains every second position

contains infinitely many *a*'s

Monadic Secondary Order Logic (MSO)

Languages of infinite words

$\underline{a}b\underline{a}a\underline{b}b\underline{a}a\underline{a}a\underline{a}b\underline{a}\ldots \in (a+b)^{\omega}$

exists a set of positions X

contains the first position

$$\cancel{\exists X} \begin{cases} \forall x\ \exists y{\leq}x \quad y{\in}X \\ \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x{\in}X \Leftrightarrow y{\notin}X) \\ \forall x\ \exists y{\geq}x \quad a(y) \wedge y{\in}X \end{cases}$$

contains every second position

$\forall z\ \exists y{>}z\ \exists_{\mathsf{fin}} X \ldots$

contains infinitely many *a*'s

Weak Monadic Secondary Order Logic (WMSO)

ω-regular languages

ω-regular languages

Automata ⟷ Logic

ω-regular languages

Automata	$\longleftrightarrow$	Logic
Muller		WMSO

Aim: find robust extensions of
ω-regular languages

Automata ⟷ Logic

Muller WMSO

Aim: find robust extensions of ω-regular languages

Automata		Logic
Muller	$\longleftrightarrow$	WMSO
max-automata		WMSO+U

Aim: find robust extensions of
ω-regular languages

Automata		Logic
Muller	⟷	WMSO
max-automata		WMSO+U
min-automata		WMSO+R

Aim: find robust extensions of ω-regular languages

Automata	Logic
Muller	WMSO
max-automata	WMSO+U
min-automata	WMSO+R
min-max-automata	WMSO+U+R

Aim: find robust extensions of ω-regular languages

Automata	Logic
Muller	WMSO
max-automata	WMSO+U
min-automata	WMSO+R
min-max-automata	WMSO+U+R
periodicity-automata	WMSO+P

Aim: find robust extensions of
ω-regular languages

Automata		Logic
Muller		WMSO
max-automata	⟷	WMSO+U
min-automata		WMSO+R
min-max-automata		WMSO+U+R
periodicity-automata		WMSO+P
…-automata		WMSO+…

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}\min(d,e)$

acceptance condition is a boolean combination of:

$\liminf(c) = \infty$
||
"c tends to ∞"

Example. $L = \{a^{n_1}b\, a^{n_2}b\, a^{n_3}b\ldots : n_1, n_2\ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}\min(c,c);\ c{:=}\min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

c:=c+1

c:=min(d,e)

acceptance condition is a boolean combination of:

liminf(c) = ∞
||
"c tends to ∞"

Example. $L = \{a^{n_1}b\ a^{n_2}b\ a^{n_3}b\ldots : n_1, n_2\ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: *c,d,z*
-when reading *a*, do *c:=c+1*
-when reading *b,* do *d:=min(c,c); c:=min(z,z);*

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}min(d,e)$

acceptance condition is a boolean combination of:

liminf(c) = ∞
||
"c tends to ∞"

Example. $L = \{a^{n_1}b\ a^{n_2}b\ a^{n_3}b\ldots : n_1,n_2\ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}min(c,c);\ c{:=}min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c := c+1$$

$$c := min(d,e)$$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$
||
"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c:=c+1$
-when reading b, do $d:=min(c,c);\ c:=min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}min(d,e)$

acceptance condition is a boolean combination of:

$\liminf(c) = \infty$
||
"c tends to ∞"

Example. $L = \{a^{n_1}b\ a^{n_2}b\ a^{n_3}b\ldots : n_1,n_2\ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}min(c,c);\ c{:=}min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}\min(d,e)$

acceptance condition is a boolean combination of:

$\liminf(c) = \infty$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\, a^{n_2} b\, a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}\min(c,c);\ c{:=}\min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c{+}1$

$c{:=}\min(d,e)$

acceptance condition is a boolean combination of:

$\liminf(c) = \infty$

||

"c tends to ∞"

Example. $L = \{a^{n_1}b\, a^{n_2}b\, a^{n_3}b\ldots : n_1,n_2\ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: *c,d,z*
-when reading *a*, do $c{:=}c{+}1$
-when reading *b,* do $d{:=}\min(c,c);\ c{:=}\min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c\text{:=}c+1$

$c\text{:=}min(d,e)$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c\text{:=}c+1$
-when reading b, do $d\text{:=}min(c,c);\ c\text{:=}min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}min(d,e)$

acceptance condition is a boolean combination of:

$\liminf(c) = \infty$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\, a^{n_2} b\, a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}min(c,c);\ c{:=}min(z,z);$

$a\ a\ a\ b\ a\ b\ a\ a\ b \ldots$

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}min(d,e)$

acceptance condition is a boolean combination of:

liminf(c) = ∞
||
"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}min(c,c);\ c{:=}min(z,z);$

$a\ a\ a\ b\ a\ b\ a\ a\ b \ldots$

c
d
z

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c := c+1$$

$$c := min(d,e)$$

acceptance condition is a boolean combination of:

liminf(c) = ∞
||
"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: *c,d,z*
-when reading *a*, do $c := c+1$
-when reading *b,* do $d := min(c,c);\ c := min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

	a a a b a b a a b…
c	0
d	0
z	0

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c{:=}c+1$$

$$c{:=}min(d,e)$$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$

||

"c tends to ∞"

Example. $L = \{a^{n_1}b\ a^{n_2}b\ a^{n_3}b\ldots : n_1, n_2\ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}min(c,c);\ c{:=}min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

	a	a	a	b	a	b	a	a	b...
c	0	1							
d	0	0							
z	0	0							

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}min(d,e)$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$
||
"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}min(c,c);\ c{:=}min(z,z);$

	a	a	a	b	a	b	a	a	b...
c	0	1	2						
d	0	0	0						
z	0	0	0						

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c := c + 1$$

$$c := min(d,e)$$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c := c+1$
-when reading b, do $d := min(c,c);\ c := min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

	a	a	a	b	a	b	a	a	b...
c	0	1	2	3					
d	0	0	0	0					
z	0	0	0	0					

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c := c+1$$

$$c := min(d,e)$$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\, a^{n_2} b\, a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: *c,d,z*
-when reading *a*, do $c:=c+1$
-when reading *b*, do $d:=min(c,c);\ c:=min(z,z);$

	a	a	a	b	a	b	a	a	b...
c	0	1	2	3	0				
d	0	0	0	0	3				
z	0	0	0	0	0				

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c := c+1$$

$$c := \min(d,e)$$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\, a^{n_2} b\, a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c:=c+1$
-when reading b, do $d:=\min(c,c);\ c:=\min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

a a a b a b a a b…

c	0	1	2	3	0	1
d	0	0	0	0	3	3
z	0	0	0	0	0	0

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c:=c+1$

$c:=min(d,e)$

acceptance condition is a boolean combination of:

liminf(c) = ∞
||
"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: *c,d,z*
-when reading *a*, do $c:=c+1$
-when reading *b,* do $d:=min(c,c);\ c:=min(z,z);$

$a\ a\ a\ b\ a\ b\ a\ a\ b\ldots$

c	0	1	2	3	0	1	0
d	0	0	0	0	3	3	1
z	0	0	0	0	0	0	0

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c := c+1$$

$$c := \min(d,e)$$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c:=c+1$
-when reading b, do $d:=\min(c,c);\ c:=\min(z,z);$

Acceptance condition: $\neg c \wedge \neg d$

	a	a	a	b	a	b	a	a	b…
c	0	1	2	3	0	1	0	1	
d	0	0	0	0	3	3	1	1	
z	0	0	0	0	0	0	0	0	

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$$c{:=}c+1$$

$$c{:=}\min(d,e)$$

acceptance condition is a boolean combination of:

$$\liminf(c) = \infty$$

||

"c tends to ∞"

Example. $L = \{a^{n_1} b\, a^{n_2} b\, a^{n_3} b \ldots : n_1, n_2 \ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}\min(c,c);\ c{:=}\min(z,z);$

	a	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*b*...
c	0	1	2	3	0	1	0	1	2
d	0	0	0	0	3	3	1	1	1
z	0	0	0	0	0	0	0	0	0

Acceptance condition: $\neg c \wedge \neg d$

Min-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}min(d,e)$

acceptance condition is a boolean combination of:

$$\begin{array}{c}\liminf(c) = \infty \\ || \\ \text{“c tends to } \infty\text{”}\end{array}$$

Example. $L = \{a^{n_1}b\ a^{n_2}b\ a^{n_3}b\ldots : n_1,n_2\ldots$ does not converge to $\infty\}$
Min-automaton has one state and three counters: c,d,z
-when reading a, do $c{:=}c+1$
-when reading b, do $d{:=}min(c,c);\ c{:=}min(z,z);$

	a	a	a	b	a	b	a	a	b...
c	0	1	2	3	0	1	0	1	2	0
d	0	0	0	0	3	3	1	1	1	2
z	0	0	0	0	0	0	0	0	0	0

Acceptance condition: $\neg c \wedge \neg d$

Tweaking the model

Tweaking the model

- Instructions $c{:=}0$, $c{:=}d$ can be implemented into the model, as in the example

Tweaking the model

- Instructions *c:=0, c:=d* can be implemented into the model, as in the example
- One can introduce the undefined counter value $\top$
  this can be eliminated by storing in the states the info about which counters are defined

Tweaking the model

- Instructions *c:=0, c:=d* can be implemented into the model, as in the example
- One can introduce the undefined counter value $\top$
  this can be eliminated by storing in the states the info about which counters are defined
- One can introduce matrix operations on counters, which stems from the semiring structure on $\{0,1,2,\ldots, \top\}$, where *min* with respect to $0<1<2<\ldots< \top$ is addition and + is multiplication

Tweaking the model

- Instructions *c:=0, c:=d* can be implemented into the model, as in the example
- One can introduce the undefined counter value $\top$
  this can be eliminated by storing in the states the info about which counters are defined
- One can introduce matrix operations on counters, which stems from the semiring structure on $\{0,1,2,\ldots, \top\}$, where *min* with respect to $0<1<2<\ldots< \top$ is addition and + is multiplication

In Example 1, *c:=c+1* can be written as:

Tweaking the model

- Instructions *c:=0, c:=d* can be implemented into the model, as in the example
- One can introduce the undefined counter value $\top$
  this can be eliminated by storing in the states the info about which counters are defined
- One can introduce matrix operations on counters, which stems from the semiring structure on $\{0,1,2,\ldots, \top\}$, where *min* with respect to $0<1<2<\ldots< \top$ is addition and + is multiplication

In Example 1, *c:=c+1* can be written as:

$$\begin{pmatrix} c & d & z \end{pmatrix} \quad := \quad \begin{pmatrix} c & d & z \end{pmatrix} \cdot \begin{pmatrix} 1 & \top & \top \\ \top & 0 & \top \\ \top & \top & 0 \end{pmatrix}.$$

Tweaking the model

- Instructions *c:=0, c:=d* can be implemented into the model, as in the example
- One can introduce the undefined counter value $\top$
  this can be eliminated by storing in the states the info about which counters are defined
- One can introduce matrix operations on counters, which stems from the semiring structure on $\{0,1,2,\ldots,\top\}$, where *min* with respect to $0<1<2<\ldots<\top$ is addition and + is multiplication

In Example 1, *c:=c+1* can be written as:

$$\begin{pmatrix} c & d & z \end{pmatrix} \quad := \quad \begin{pmatrix} c & d & z \end{pmatrix} \cdot \begin{pmatrix} 1 & \top & \top \\ \top & 0 & \top \\ \top & \top & 0 \end{pmatrix}.$$

d:=min(c,c); c:=z can be written as:

Tweaking the model

- Instructions *c:=0, c:=d* can be implemented into the model, as in the example
- One can introduce the undefined counter value $\top$
  this can be eliminated by storing in the states the info about which counters are defined
- One can introduce matrix operations on counters, which stems from the semiring structure on $\{0,1,2,\ldots, \top\}$, where *min* with respect to $0<1<2<\ldots< \top$ is addition and + is multiplication

In Example 1, *c:=c+1* can be written as:

$$\begin{pmatrix} c & d & z \end{pmatrix} \quad := \quad \begin{pmatrix} c & d & z \end{pmatrix} \cdot \begin{pmatrix} 1 & \top & \top \\ \top & 0 & \top \\ \top & \top & 0 \end{pmatrix}.$$

d:=min(c,c); c:=z can be written as:

$$\begin{pmatrix} c & d & z \end{pmatrix} \quad := \quad \begin{pmatrix} c & d & z \end{pmatrix} \cdot \begin{pmatrix} \top & 0 & \top \\ \top & \top & \top \\ 0 & \top & 0 \end{pmatrix}.$$

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts a's on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c := c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1) = (0, \top)$.

$$
\begin{array}{llll}
a: & \begin{pmatrix} c_0 & c_1 \end{pmatrix} & := & \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}. \\
b: & \begin{pmatrix} c_0 & c_1 \end{pmatrix} & := & \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.
\end{array}
$$

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c{:=}c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

a a a b b b a a b...

c_0

c_1

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c{:=}c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

$a\ a\ a\ b\ b\ b\ a\ a\ b\ldots$

c_0 0

c_1 $\top$

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c{:=}c{+}1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

a a a b b b a a b...

c_0	0	$\top$
c_1	$\top$	1

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c{:=}c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \qquad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \quad := \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \qquad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \quad := \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

a a a b b b a a b...

c_0	0	$\top$	1
c_1	$\top$	1	$\top$

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts a's on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c{:=}c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \qquad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \quad := \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \qquad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \quad := \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

$a\ a\ a\ b\ b\ b\ a\ a\ b\ldots$

$c_0 \quad 0 \ \top \ 1 \ \top$

$c_1 \quad \top \ 1 \ \top \ 2$

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

a a a b b b a a b...

c_0 0 ⊤ 1 ⊤ 2

c_1 ⊤ 1 ⊤ 2 ⊤

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$
\begin{aligned}
a: \quad & \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}. \\
b: \quad & \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.
\end{aligned}
$$

a a a b b b a a b...

c_0	0	$\top$	1	$\top$	2	$\top$
c_1	$\top$	1	$\top$	2	$\top$	2

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

a a a b b b a a b...

c_0	0	$\top$	1	$\top$	2	$\top$	2	
c_1	$\top$	1	$\top$	2	$\top$	2	$\top$	

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c := c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1) = (0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

a a a b b b a a b...

c_0	0	$\top$	1	$\top$	2	$\top$	2	$\top$	
c_1	$\top$	1	$\top$	2	$\top$	2	$\top$	3	

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c := c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1) = (0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

$a\ a\ a\ b\ b\ b\ a\ a\ b\ldots$

c_0 0 ⊤ 1 ⊤ 2 ⊤ 2 ⊤ 3

c_1 ⊤ 1 ⊤ 2 ⊤ 2 ⊤ 3 ⊤

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.

Has states q_0, q_1 and one counter c.

Transitions:

-saw a in state q_0 – go to q_1; $c:=c+1$

-saw a in state q_1 – go to q_0

-saw b in state q_0 – go to q_1

-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.

The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} := \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

		a		*a*		*a*		*b*		*b*		*b*		*a*		*a*		*b...*
c_0	0		$\top$		1		$\top$		2		$\top$		2		$\top$		3		$\top$
c_1	$\top$		1		$\top$		2		$\top$		2		$\top$		3		$\top$		3

Theorem. Min-automata are equivalent to min-automata in matrix form, with one state.

Proof. We eliminate states as in the following example.

Example. Min-automaton which counts *a*'s on odd positions.
Has states q_0, q_1 and one counter c.
Transitions:
-saw a in state q_0 – go to q_1; $c{:=}c+1$
-saw a in state q_1 – go to q_0
-saw b in state q_0 – go to q_1
-saw b in state q_1 – go to q_0

Min-automaton in matrix form with one state and two counters: c_0, c_1.
The initial counter valuation is $(c_0, c_1)=(0, \top)$.

$$a: \qquad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \quad := \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 1 & \top \end{pmatrix}.$$

$$b: \qquad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \quad := \quad \begin{pmatrix} c_0 & c_1 \end{pmatrix} \cdot \begin{pmatrix} \top & 0 \\ 0 & \top \end{pmatrix}.$$

	a	*a*	*a*	*b*	*b*	*b*	*a*	*a*	*b*...
c_0	0	$\top$	1	$\top$	2	$\top$	2	$\top$	3	$\top$
c_1	$\top$	1	$\top$	2	$\top$	2	$\top$	3	$\top$	3

In the other direction, one can convert a min-automaton in matrix form by simulating a matrix operation as a sequence of counter operations, and then eliminating $\top$ values by storing them in the state.

Nondeterministic min-automata

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L = \{a^{n_1}\, b\ a^{n_2}\, b\ a^{n_3}\, b \ldots : n_1, n_2 \ldots$ is unbounded$\}$.

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\,b\;a^{n2}\,b\;a^{n3}\,b\ldots: n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following

Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b\ldots : n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following

Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;\ q$ is the "skip block" state
counters: c,d,z

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\ a^{n2}\, b\ a^{n3}\, b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:
states: *p,q; q* is the "skip block" state
counters: c,d,z

transitions:
saw *b* in state *p* - go to *p* or *q; d:=c; c:=z;*
saw *b* in state *q* - go to *p* or *q*
saw *a* in state *p* - go to *p; c:=c+1;*
saw *a* in state *q* - go to *q;*

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;\ q$ is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d{:=}c;\ c{:=}z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c{:=}c+1;$
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state
c
d

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2 \ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p, q; q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d := c$; $c := z$;
saw b in state q - go to p or q
saw a in state p - go to p; $c := c+1$;
saw a in state q - go to q;

	a b a a a b a b a a a a b a b…
state	p
c	0
d	0

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;$ q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d{:=}c;\ c{:=}z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c{:=}c+1;$
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state	p	p
c	0	1
d	0	0

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b\ldots: n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;$ q is the "skip block" state
counters: c,d,z

	a	*b*	*a*	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*a*	*a*	*b*	*a*	*b*...
state	*p*	*p*	*p*												
c	0	1	0												
d	0	0	1												

transitions:
saw b in state p - go to p or q; $d:=c;\ c:=z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c:=c+1;$
saw a in state q - go to q;

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;$ q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d{:=}c;$ $c{:=}z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c{:=}c+1;$
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state	p	p	p	p
c	0	1	0	1
d	0	0	1	1

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L = \{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b \ldots : n_1, n_2 \ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2 \ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p, q; q is the "skip block" state
counters: c,d,z

	a	b	a	a	a	b	a	b	a	a	a	a	b	a	$b \ldots$
state	p	p	p	p	p										
c	0	1	0	1	2										
d	0	0	1	1	1										

transitions:
saw b in state p - go to p or q; $d{:=}c$; $c{:=}z$;
saw b in state q - go to p or q
saw a in state p - go to p; $c{:=}c+1$;
saw a in state q - go to q;

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p,q; q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d{:=}c$; $c{:=}z$;
saw b in state q - go to p or q
saw a in state p - go to p; $c{:=}c+1$;
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state	p	p	p	p	p	p
c	0	1	0	1	2	3
d	0	0	1	1	1	1

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\, a^{n_2} b\, a^{n_3} b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following

Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p, q; q is the "skip block" state

counters: c,d,z

transitions:

saw b in state p - go to p or q; $d{:=}c$; $c{:=}z$;

saw b in state q - go to p or q

saw a in state p - go to p; $c{:=}c+1$;

saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state	p	p	p	p	p	p	q
c	0	1	0	1	2	3	0
d	0	0	1	1	1	1	3

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}b\ a^{n2}b\ a^{n3}b\ldots : n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;$ q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d{:=}c;\ c{:=}z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c{:=}c{+}1;$
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state	p	p	p	p	p	p	q	q
c	0	1	0	1	2	3	0	0
d	0	0	1	1	1	1	3	3

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1}b\ a^{n_2}b\ a^{n_3}b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p,q; q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d:=c$; $c:=z$;
saw b in state q - go to p or q
saw a in state p - go to p; $c:=c+1$;
saw a in state q - go to q;

	$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$
state	$p\ p\ p\ p\ p\ p\ q\ q\ p$
c	0 1 0 1 2 3 0 0 0
d	0 0 1 1 1 1 3 3 3

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2 \ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: *p,q; q* is the "skip block" state
counters: c,d,z

transitions:
saw *b* in state *p* - go to *p* or *q; d:=c; c:=z;*
saw *b* in state *q* - go to *p* or *q*
saw *a* in state *p* - go to *p; c:=c+1;*
saw *a* in state *q* - go to *q;*

a b a a a b a b a a a a b a b…

state	*p*	*p*	*p*	*p*	*p*	*p*	*q*	*q*	*p*	*p*
c	0	1	0	1	2	3	0	0	0	1
d	0	0	1	1	1	1	3	3	3	3

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2 \ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p,q; q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d:=c$; $c:=z$;
saw b in state q - go to p or q
saw a in state p - go to p; $c:=c+1$;
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b \ldots$

state	p	p	p	p	p	p	q	q	p	p	p
c	0	1	0	1	2	3	0	0	0	1	2
d	0	0	1	1	1	1	3	3	3	3	3

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p,q; q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d:=c;\ c:=z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c:=c+1;$
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state	$p\ p\ p\ p\ p\ p\ q\ q\ p\ p\ p\ p$
c	0 1 0 1 2 3 0 0 0 1 2 3
d	0 0 1 1 1 1 3 3 3 3 3 3

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;$ q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d{:=}c;\ c{:=}z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c{:=}c+1;$
saw a in state q - go to q;

$a\ b\ a\ a\ a\ b\ a\ b\ a\ a\ a\ a\ b\ a\ b\ldots$

state	p	p	p	p	p	p	q	q	p	p	p	p	p
c	0	1	0	1	2	3	0	0	0	1	2	3	4
d	0	0	1	1	1	1	3	3	3	3	3	3	3

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\,b\;a^{n2}\,b\;a^{n3}\,b\ldots: n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: $p,q;\ q$ is the “skip block” state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d:=c;\ c:=z;$
saw b in state q - go to p or q
saw a in state p - go to p; $c:=c+1;$
saw a in state q - go to q;

	a	b	a	a	a	b	a	b	a	a	a	a	b	a	$b\ldots$
state	p	p	p	p	p	p	q	q	p	p	p	p	p	q	
c	0	1	0	1	2	3	0	0	0	1	2	3	4	0	
d	0	0	1	1	1	1	3	3	3	3	3	3	3	4	

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L = \{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b \ldots : n_1, n_2 \ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2 \ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p,q; q is the "skip block" state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d:=c$; $c:=z$;
saw b in state q - go to p or q
saw a in state p - go to p; $c:=c+1$;
saw a in state q - go to q;

	a	b	a	a	a	b	a	b	a	a	a	a	b	a	$b\ldots$
state	p	p	p	p	p	p	q	q	p	p	p	p	p	q	q
c	0	1	0	1	2	3	0	0	0	1	2	3	4	0	0
d	0	0	1	1	1	1	3	3	3	3	3	3	3	4	4

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n_1}b\ a^{n_2}b\ a^{n_3}b\ldots : n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

states: p,q; q is the “skip block” state
counters: c,d,z

transitions:
saw b in state p - go to p or q; $d:=c$; $c:=z$;
saw b in state q - go to p or q
saw a in state p - go to p; $c:=c+1$;
saw a in state q - go to q;

	a	b	a	a	a	b	a	b	a	a	a	a	b	a	$b\ldots$	
state	p	p	p	p	p	p	q	q	p	p	p	p	p	q	q	q
c	0	1	0	1	2	3	0	0	0	1	2	3	4	0	0	0
d	0	0	1	1	1	1	3	3	3	3	3	3	3	4	4	4

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\ a^{n2}\, b\ a^{n3}\, b\ldots: n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b\ldots : n_1,n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1,n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

Theorem. A deterministic min-automaton cannot recognize the language L.

Nondeterministic min-automata

Nondeterministic min-automata are strictly more expressive than deterministic ones. Separating language:

$L=\{a^{n1}\, b\ a^{n2}\, b\ a^{n3}\, b\ldots : n_1, n_2\ldots$ is unbounded$\}$.

Can be recognized by a nondeterministic min-automaton, due to the following
Observation. The sequence $n_1, n_2\ldots$ is unbounded iff it contains a subsequence which tends to ∞.

A nondeterministic automaton can guess the subsequence:

Theorem. A deterministic min-automaton cannot recognize the language *L.*

Corollary. Deterministic min-automaton are not closed under the second order existential quantifier $\exists X$.

Max-automata

deterministic automata with counters
transitions invoke counter operations:

c:=c+1

c:=max(d,e)

acceptance condition is a boolean combination of:

limsup(c) = ∞
||
"c has unbounded values"

Example. $L = \{a^{n1} b\ a^{n2} b\ a^{n3} b\ldots : n_1, n_2\ldots$ is unbounded$\}$

Theorem. Min-automata and max-automata have incomparable expressiveness.

Min-max-automata –

boolean combinations of min- and max-automata.

Max-automata

deterministic automata with counters
transitions invoke counter operations:

$c{:=}c+1$

$c{:=}max(d,e)$

acceptance condition is a boolean combination of:

limsup(c) = ∞
||
"c has unbounded values"

Example. $L = \{a^{n1}\, b\; a^{n2}\, b\; a^{n3}\, b\ldots : n_1, n_2 \ldots$ is unbounded$\}$

Theorem. Min-automata and max-automata have incomparable expressiveness.

Min-max-automata –
boolean combinations of min- and max-automata.

Max-automata

deterministic automata with counters
transitions invoke counter operations:

$$c{:=}c+1$$

$$c{:=}max(d,e)$$

acceptance condition is a boolean combination of:

$$\limsup(c) = \infty$$

||

“c has unbounded values”

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots \text{ is unbounded}\}$

Theorem. Min-automata and max-automata have incomparable expressiveness.

Min-max-automata –
boolean combinations of min- and max-automata.

Max-automata

deterministic automata with counters
transitions invoke counter operations:

$$c{:=}c+1$$

$$c{:=}max(d,e)$$

acceptance condition is a boolean combination of:

limsup(c) = ∞
||
"c has unbounded values"

Example. $L = \{a^{n1} b\ a^{n2} b\ a^{n3} b \ldots : n_1, n_2 \ldots$ is unbounded$\}$

Theorem. Min-automata and max-automata have incomparable expressiveness.

Min-max-automata –
boolean combinations of min- and max-automata.

Max-automata

deterministic automata with counters
transitions invoke counter operations:

$$c := c+1$$

$$c := max(d,e)$$

acceptance condition is a boolean combination of:

$$\limsup(c) = \infty$$

||

"c has unbounded values"

Example. $L = \{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1, n_2 \ldots$ is unbounded$\}$

Theorem. Min-automata and max-automata have incomparable expressiveness.

Min-max-automata –

boolean combinations of min- and max-automata.

Emptiness of min-max-automata

Emptiness of min-max-automata

Theorem. There exists an algorithm deciding emptiness of min-max-automata, which runs in polynomial space.

Emptiness of min-max-automata

Theorem. There exists an algorithm deciding emptiness of min-max-automata, which runs in polynomial space.

Proof. min-max-automata are a special case of ωBS-automata (Bojańczyk, Colcombet [06]), so emptiness is decidable. This gives bad complexity, however.

Emptiness of min-max-automata

Theorem. There exists an algorithm deciding emptiness of min-max-automata, which runs in polynomial space.

Proof. min-max-automata are a special case of ωBS-automata (Bojańczyk, Colcombet [06]), so emptiness is decidable. This gives bad complexity, however.

Another proof. Uses profinite and semigroup methods.
Is related to:

- Limitedness problem for Distance Automata – Hashiguchi [82], Leung [91], Simon [94], Kirsten [05], Colcombet [09]
- Semiring of matrices over the tropical semiring

Emptiness of min-max-automata

Theorem. There exists an algorithm deciding emptiness of min-max-automata, which runs in polynomial space.

Proof. min-max-automata are a special case of ωBS-automata (Bojańczyk, Colcombet [06]), so emptiness is decidable. This gives bad complexity, however.

Another proof. Uses profinite and semigroup methods.
Is related to:

- Limitedness problem for Distance Automata – Hashiguchi [82], Leung [91], Simon [94], Kirsten [05], Colcombet [09]
- Semiring of matrices over the tropical semiring

Theorem. Emptiness of min- and max-automata is PSPACE-hard.

Emptiness of min-max-automata

Theorem. There exists an algorithm deciding emptiness of min-max-automata, which runs in polynomial space.

Proof. min-max-automata are a special case of ωBS-automata (Bojańczyk, Colcombet [06]), so emptiness is decidable. This gives bad complexity, however.

Another proof. Uses profinite and semigroup methods.
Is related to:

– Limitedness problem for Distance Automata – Hashiguchi [82], Leung [91], Simon [94], Kirsten [05], Colcombet [09]
– Semiring of matrices over the tropical semiring

Theorem. Emptiness of min- and max-automata is PSPACE-hard.

Proof. Standard reduction from universality of nondeterministic finite automata.

Logic

Max-automata

Logic

Max-automata

Extension of WMSO by the quantifier

Logic

Max-automata

Extension of WMSO by the quantifier

$$\mathsf{U}X \quad \varphi(X)$$

which says

„there exist arbitrarily large (finite) sets X, satisfying $\varphi(X)$”

Logic

Max-automata

Extension of WMSO by the quantifier

$$\mathsf{U}X \;\; \varphi(X)$$

which says

„there exist arbitrarily large (finite) sets X, satisfying $\varphi(X)$”

Language: $\{a^{n_1} b\, a^{n_2} b\, a^{n_3} b \ldots : n_1\, n_2\, n_3 \ldots$ is unbounded$\}$

Logic

Max-automata

Extension of WMSO by the quantifier

$$\mathsf{U}X \;\; \varphi(X)$$

which says

„there exist arbitrarily large (finite) sets X, satisfying $\varphi(X)$"

Language: $\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1\, n_2\, n_3 \ldots$ is unbounded$\}$

$\mathsf{U}X$ *"X is a block of a's"*

Logic

Max-automata $\longleftrightarrow$ Min-automata

Extension of WMSO by the quantifier

$$\mathsf{U}X \;\; \varphi(X)$$

which says

„*there exist arbitrarily large (finite) sets X, satisfying* $\varphi(X)$"

Language: $\{a^{n_1} b\; a^{n_2} b\; a^{n_3} b\ldots : n_1\, n_2\, n_3\ldots$ is unbounded$\}$

$\mathsf{U}X$ *"X is a block of a's"*

Logic

Max-automata $\longleftrightarrow$ Min-automata

Extension of WMSO by the quantifier

$\mathsf{U}X \;\; \varphi(X)$ $\longleftrightarrow$ $\mathsf{R}X \;\; \varphi(X)$

which says

„there exist arbitrarily large (finite) sets X, satisfying $\varphi(X)$"

Language: $\{a^{n_1} b \; a^{n_2} b \; a^{n_3} b\ldots : n_1 \, n_2 \, n_3 \ldots$ is unbounded$\}$

$\mathsf{U}X$ *"X is a block of a's"*

Logic

Max-automata $\longleftrightarrow$ Min-automata

Extension of WMSO by the quantifier

$\mathsf{U}X\ \ \varphi(X) \longleftrightarrow \mathsf{R}X\ \ \varphi(X)$

which says

„there exist infinitely many sets X of bounded size, satisfying $\varphi(X)$"

Language: $\{a^{n_1}\, b\ a^{n_2}\, b\ a^{n_3}\, b\ldots : n_1\, n_2\, n_3 \ldots$ is unbounded$\}$

$\mathsf{U}X$ *"X is a block of a's"*

Logic

Max-automata $\longleftrightarrow$ Min-automata

Extension of WMSO by the quantifier

$\mathsf{U}X \ \ \varphi(X)$ $\longleftrightarrow$ $\mathsf{R}X \ \ \varphi(X)$

which says

„*there exist infinitely many sets X of bounded size, satisfying* $\varphi(X)$"

Language: $\{a^{n_1} b \ a^{n_2} b \ a^{n_3} b \ldots : n_1 \, n_2 \, n_3 \ldots$ converges to $\infty\}$

$\mathsf{U}X$ *"X is a block of a's"*

Logic

Max-automata $\longleftrightarrow$ Min-automata

Extension of WMSO by the quantifier

$\mathsf{U}X\ \ \varphi(X)$ $\longleftrightarrow$ $\mathsf{R}X\ \ \varphi(X)$

which says

„there exist infinitely many sets X of bounded size, satisfying $\varphi(X)$"

Language: $\{a^{n_1} b\ a^{n_2} b\ a^{n_3} b \ldots : n_1\, n_2\, n_3 \ldots$ converges to $\infty\}$

$\neg\mathsf{R}X$ *"X is a block of a's"*

$\mathsf{R}X\ \varphi(X)$: „*there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$"

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

$\mathsf{R}X\ \varphi(X)$: *„there exist infinitely many sets X of the same size, satisfying $\varphi(X)$"*

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

$\mathsf{R}X\ \ \varphi(X)$: *„there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$*"*

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

values of c do not tend to ∞

$\mathsf{R}X\ \varphi(X)$: *„there exist infinitely many sets X of the same size, satisfying $\varphi(X)$"*

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

values of c do not tend to ∞

d:=d+1; c:=min(d,e); c:=c+1 ...

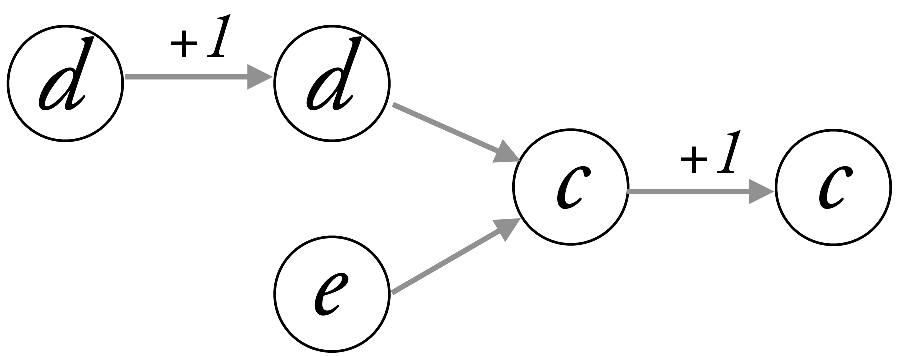
$\mathsf{R}X\ \varphi(X)$: *„there exist infinitely many sets X of the same size, satisfying $\varphi(X)$"*

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

values of c do not tend to ∞

d:=d+1; c:=min(d,e); c:=c+1 ...



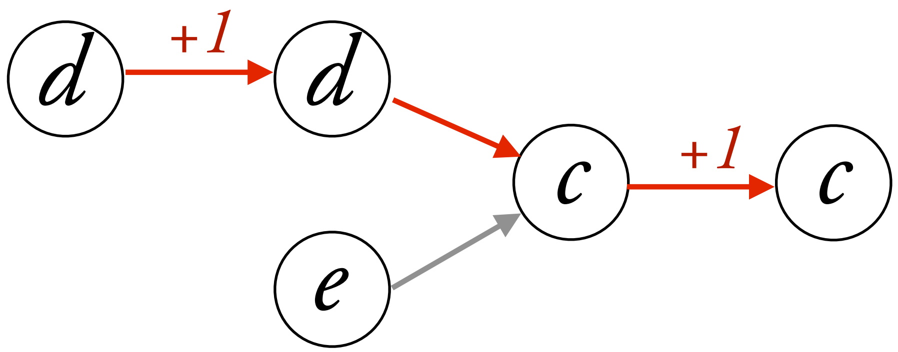
RX $\varphi(X)$: „*there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$”

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

values of c do not tend to ∞

d:=d+1; c:=min(d,e); c:=c+1 ...



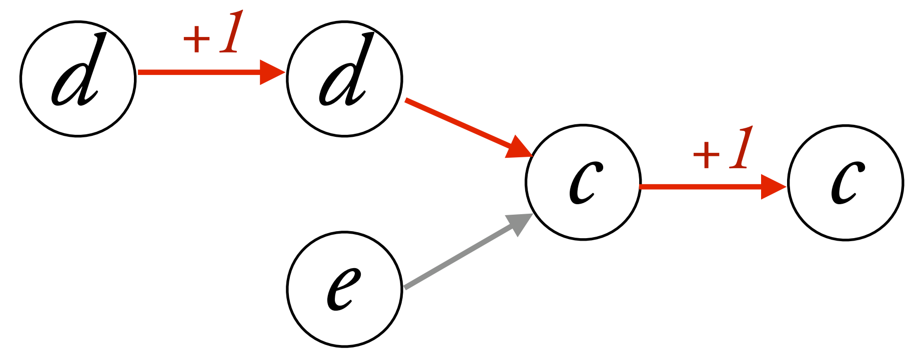
$\mathsf{R}X\ \varphi(X)$: „*there exist infinitely many sets* X *of the same size, satisfying* $\varphi(X)$”

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

values of c do not tend to ∞

d:=d+1; c:=min(d,e); c:=c+1 …



R X. *there is a path starting in an initial counter, ending in c, with increments at positions in the set* X

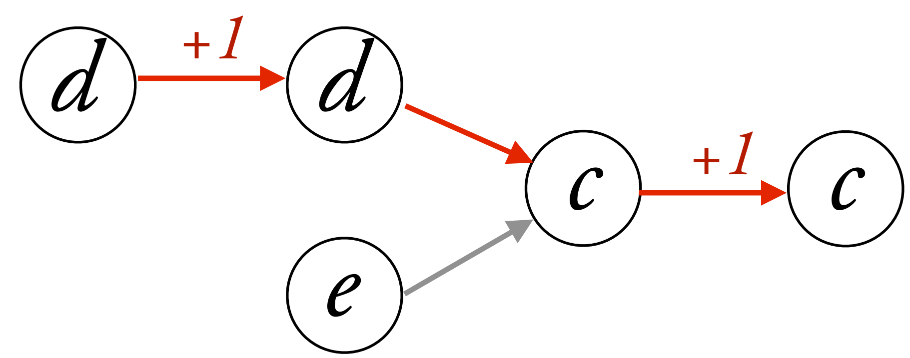
RX $\varphi(X)$: „*there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$"

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

values of c do not tend to ∞

d:=d+1; c:=min(d,e); c:=c+1 …



R X. *there is a path starting in an initial counter, ending in c, with increments at positions in the set X*

In particular, min-automata recognize boolean combinations of languages of the form RX $\varphi(X)$, where $\varphi(X)$ is WMSO and such that if $w,X \models \varphi$, then there is a prefix v of w such that $vu,X \models \varphi$ for any suffix u. We call RX $\varphi(X)$ a *prefix* R*-formula.*

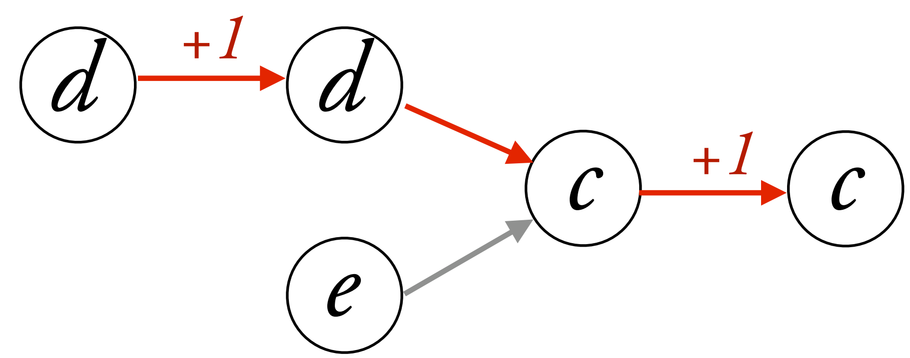
RX $\varphi(X)$: *„there exist infinitely many sets X of the same size, satisfying $\varphi(X)$"*

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Proof (easy direction). The acceptance condition is a boolean combination of conditions:

values of c do not tend to ∞

d:=d+1; c:=min(d,e); c:=c+1 ...



R X. *there is a path starting in an initial counter, ending in c, with increments at positions in the set X*

In particular, min-automata recognize boolean combinations of languages of the form RX $\varphi(X)$, where $\varphi(X)$ is WMSO and such that if $w,X \vDash \varphi$, then there is a prefix v of w such that $vu,X \vDash \varphi$ for any suffix u. We call RX $\varphi(X)$ a *prefix* R*-formula.*

(harder direction). Construct automaton by induction on structure of formula.

For deterministic automata, closure under boolean operations is for free. Must show closure under $\exists_{\text{fin}}$ and that nested R quantifiers can be denested. Follows from a more general theorem.

WMSO + U ⟷ WMSO + R

max-automata ⟷ min-automata

$$\begin{array}{ccc} \text{WMSO + U} & \longleftrightarrow & \text{WMSO + R} \\ \| & & \\ \text{max-automata} & \longleftrightarrow & \text{min-automata} \end{array}$$

Theorem. WMSO+U has the same expressive power as deterministic max-automata.

$$\begin{array}{ccc} \text{WMSO} + \mathsf{U} & \longleftrightarrow & \text{WMSO} + \mathsf{R} \\ \| & & \| \\ \text{max-automata} & \longleftrightarrow & \text{min-automata} \end{array}$$

Theorem. WMSO+U has the same expressive power as deterministic max-automata.

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

WMSO + U + R

||

max-automata min-automata

Theorem. WMSO+U has the same expressive power as deterministic max-automata.

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

What if we allow both U and R?

WMSO + U + R

$\parallel$

max-automata min-automata

Theorem. WMSO+U has the same expressive power as deterministic max-automata.

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Theorem. WMSO+U+R has the same expressive power as boolean combinations of min- and max-automata.

WMSO + U + R

$\parallel$

min-max-automata

Theorem. WMSO+U has the same expressive power as deterministic max-automata.

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Theorem. WMSO+U+R has the same expressive power as boolean combinations of min- and max-automata.

WMSO + U + R

||

min-max-automata

Theorem. WMSO+U has the same expressive power as deterministic max-automata.

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Theorem. WMSO+U+R has the same expressive power as boolean combinations of min- and max-automata.

Equivalently: Nesting the quantifiers U and R does not contribute anything to the expressive power of WMSO.

WMSO + U + R

||

min-max-automata

Theorem. WMSO+U has the same expressive power as deterministic max-automata.

Theorem. WMSO+R has the same expressive power as deterministic min-automata.

Theorem. WMSO+U+R has the same expressive power as boolean combinations of min- and max-automata.

Equivalently: Nesting the quantifiers U and R does not contribute anything to the expressive power of WMSO.
Follows from the more general theorem.

Periodicity-automata

Periodicity-automata

Deterministic automata allowed to verify that certain states appear in an ultimately periodic way

Periodicity-automata

Deterministic automata allowed to verify that certain states
appear in an ultimately periodic way

WMSO + P

Periodicity-automata

Deterministic automata allowed to verify that certain states appear in an ultimately periodic way

WMSO + P

Extension of WMSO by the following quantifier

Periodicity-automata

Deterministic automata allowed to verify that certain states appear in an ultimately periodic way

WMSO + P

Extension of WMSO by the following quantifier

$$\mathsf{P}x \;\; \varphi(x)$$

"*the set of positions x satisfying $\varphi(x)$ is ultimately periodic*"

Periodicity-automata

$$\|$$

WMSO + P

Theorem. WMSO + P has the same expressive power as periodicity-automata.

Periodicity-automata

$$\parallel$$

WMSO + P

Theorem. WMSO + P has the same expressive power as periodicity-automata.

Theorem. Emptiness of periodicity automata is decidable. Therefore, WMSO + P has decidable satisfiability.

Periodicity-automata

$\|$

WMSO + P

Theorem. WMSO + P has the same expressive power as periodicity-automata.

Theorem. Emptiness of periodicity automata is decidable. Therefore, WMSO + P has decidable satisfiability.

Theorem. WMSO + R + U + P has the same expressive power as boolean combinations of min- max- and periodicity-automata.

General framework

General framework

Theorem. A WMSO+Q_1+Q_2+…+Q_n formula is equivalent to a boolean combination of formulas of the form $\mathsf{Q}_k X \ \varphi_k(X)$.
(We require some additional conditions on the quantifiers $\mathsf{Q}_1, \mathsf{Q}_2, \ldots, \mathsf{Q}_n$ which will be phrased later)

Another view on min- and max-automata

Another view on min- and max-automata

A min-automaton can be viewed as:

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f\colon A \to B$ which outputs a sequence of counter operations

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \to B$ which outputs a sequence of counter operations

$a \quad b \quad b \quad a \quad b. \ldots\ldots$

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \to B$ which outputs a sequence of counter operations

$a \qquad b \qquad b \qquad a \qquad b. \ldots\ldots$

$d{:=}d+1;$

Another view on
min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f\colon A \to B$
which outputs a sequence of counter operations

$a \qquad\qquad b \qquad\qquad b \qquad\qquad a \qquad\qquad b\ldots\ldots$

$d{:=}d+1; \quad c{:=}min(d,e);$

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \to B$ which outputs a sequence of counter operations

a b b a b. . . . . .

$d:=d+1;$ $c:=min(d,e);$ $c:=c+1;$ $d:=d+1;$

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f\colon A \rightarrow B$
which outputs a sequence of counter operations

$a \quad b \quad b \quad a \quad b\ldots\ldots$

$d{:=}d+1; \quad c{:=}min(d,e); \quad c{:=}c+1;\ d{:=}d+1; \quad c{:=}c+1;$

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \to B$ which outputs a sequence of counter operations

$a \quad b \quad b \quad a \quad b. \ldots\ldots$

$d{:=}d{+}1; \quad c{:=}min(d,e); \quad c{:=}c{+}1;\ d{:=}d{+}1; \quad c{:=}c{+}1; \quad c{:=}min(c,c); \ldots\ldots$

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \rightarrow B$ which outputs a sequence of counter operations

$$\begin{array}{llllll} a & b & b & a & b\ldots\ldots \\ d{:=}d{+}1; & c{:=}min(d,e); & c{:=}c{+}1;\ d{:=}d{+}1; & c{:=}c{+}1; & c{:=}min(c,c);\ldots\ldots \end{array}$$

We consider the language $F \subseteq B^\omega$ of sequences of instructions in which the appropriate counters converge to ∞.

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \to B$ which outputs a sequence of counter operations

$$\begin{array}{ccccc} a & b & b & a & b. \ldots\ldots \\ d{:=}d{+}1; & c{:=}min(d,e); & c{:=}c{+}1;\ d{:=}d{+}1; & c{:=}c{+}1; & c{:=}min(c,c);\ldots\ldots \end{array}$$

We consider the language $F \subseteq B^\omega$ of sequences of instructions in which the appropriate counters converge to ∞.

The language F is *prefix-independent,* i.e. $F{=}B^*F$.

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f\colon A \to B$ which outputs a sequence of counter operations

$$
\begin{array}{cccccc}
a & b & b & a & b. \ldots\ldots \\
d{:=}d{+}1; & c{:=}min(d,e); & c{:=}c{+}1;\ d{:=}d{+}1; & c{:=}c{+}1; & c{:=}min(c,c);. \ldots\ldots
\end{array}
$$

We consider the language $F \subseteq B^\omega$ of sequences of instructions in which the appropriate counters converge to ∞.

The language F is *prefix-independent*, i.e. $F = B^* F$.

The automaton accepts a word $w \in A^\omega$ iff $f(w) \in F$.

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \rightarrow B$
which outputs a sequence of counter operations

An *F-automaton*

We consider the language $F \subseteq B^\omega$ of sequences of instructions in which the appropriate counters converge to ∞.

The language F is *prefix-independent*, i.e. $F = B^* F$.

The automaton accepts a word $w \in A^\omega$ iff $f(w) \in F$.

Another view on min- and max-automata

A min-automaton can be viewed as:

A deterministic letter-to-letter transducer $f: A \to B$
which outputs a sequence of counter operations

An *F-automaton*

We consider the language $F \subseteq B^\omega$ of sequences of instructions in which the appropriate counters converge to ∞.

The language F is *prefix-independent*, i.e. $F = B^* F$.

The automaton accepts a word $w \in A^\omega$ iff $f(w) \in F$.

Similarly, Büchi, Muller, parity, max- automata are *F*-automata

Another view on quantifiers U, R, $\exists_{\mathsf{fin}}$

Another view on quantifiers U, R, $\exists_{fin}$

They speak about properties of families of finite sets of positions:

Another view on quantifiers U, R, $\exists_{\mathrm{fin}}$

They speak about properties of families of finite sets of positions:

$\mathsf{U}X \;\; \varphi(X)$ "*there exist arbitrarily large (finite) sets X, satisfying* $\varphi(X)$"

Another view on quantifiers U, R, $\exists_{\text{fin}}$

They speak about properties of families of finite sets of positions:

$\mathsf{U}X \ \ \varphi(X)$ "*there exist arbitrarily large (finite) sets X, satisfying* $\varphi(X)$"

$\mathsf{R}X \ \ \varphi(X)$ "*there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$"

Another view on quantifiers U, R, $\exists_{\text{fin}}$

They speak about properties of families of finite sets of positions:

$\mathsf{U}X \ \ \varphi(X)$ "*there exist arbitrarily large (finite) sets X, satisfying* $\varphi(X)$"

$\mathsf{R}X \ \ \varphi(X)$ "*there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$"

$\exists_{\text{fin}} X \ \ \varphi(X)$ "*the family of finite sets X which satisfy* $\varphi(X)$ *is nonempty*"

Another view on quantifiers U, R, $\exists_{\text{fin}}$

They speak about properties of families of finite sets of positions:

$\mathsf{U}X \ \ \varphi(X)$ "*there exist arbitrarily large (finite) sets X, satisfying* $\varphi(X)$"

$\mathsf{R}X \ \ \varphi(X)$ "*there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$"

$\exists_{\text{fin}} X \ \ \varphi(X)$ "*the family of finite sets X which satisfy* $\varphi(X)$ *is nonempty*"

$\mathsf{Q}X \ \ \varphi(X)$ "*the family of finite sets X satisfying* $\varphi(X)$ *has a property* Q"

Another view on quantifiers U, R, $\exists_{\mathsf{fin}}$

They speak about properties of families of finite sets of positions:

$\mathsf{U}X \ \ \varphi(X)$ "*there exist arbitrarily large (finite) sets X, satisfying* $\varphi(X)$"

$\mathsf{R}X \ \ \varphi(X)$ "*there exist infinitely many sets X of the same size, satisfying* $\varphi(X)$"

$\exists_{\mathsf{fin}} X \ \ \varphi(X)$ "*the family of finite sets X which satisfy* $\varphi(X)$ *is nonempty*"

$\mathsf{Q}X \ \ \varphi(X)$ "*the family of finite sets X satisfying* $\varphi(X)$ *has a property* Q"

A locus quantifier: any property Q of families of finite sets of positions

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\, L = \{w \in A^{\omega}\colon\ \mathsf{Q}X\, [w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\, L = \{w \in A^\omega : \ \mathsf{Q}X\, [w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\, L = \{w \in A^\omega : \ \mathsf{Q}X\, [w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x \ \forall y \quad \mathrm{suc}(x,y) \Rightarrow \ (x \in X \ \Leftrightarrow y \notin X)$

X - a finite set of positions; $\qquad w, X \vDash \varphi$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\, L = \{w \in A^{\omega} : \ \mathsf{Q} X\, [w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\qquad w,X \vDash \varphi$

$\mathscr{X}$ - a family of sets X; $\qquad \mathscr{X}_\varphi = \{X : w,X \vDash \varphi\}$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w,X \vDash \varphi$

$\mathscr{X}$ - a family of sets X; $\quad \mathscr{X}_\varphi = \{X : w,X \vDash \varphi\}$

Q - a property of sets $\mathscr{X}$; $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathscr{X}_\varphi \in \mathsf{Q}$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w,X \vDash \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X : w,X \vDash \varphi\}$

Q - a property of sets $\mathcal{X}$; $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega :\ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\qquad w,X \vDash \varphi$

$\mathcal{X}$ - a family of sets X; $\qquad \mathcal{X}_\varphi = \{X : w,X \vDash \varphi\}$

Q - a property of sets $\mathcal{X}$; $\qquad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathsf{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\qquad w,X \models \varphi$

$\mathcal{X}$ - a family of sets X; $\qquad \mathcal{X}_\varphi = \{X : w,X \models \varphi\}$

Q - a property of sets $\mathcal{X}$; $\qquad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathsf{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets* X *of same size*$\}$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w,X \vDash \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X : w,X \vDash \varphi\}$

Q - a property of sets $\mathcal{X}$; $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathsf{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets* X *of same size*$\}$

$\mathsf{U} = \{\mathcal{X} : \mathcal{X}$ *contains sets* X *of arbitrarily large size*$\}$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w, X \models \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X : w, X \models \varphi\}$

Q - a property of sets $\mathcal{X}$; $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathsf{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets* X *of same size*$\}$

$\mathsf{U} = \{\mathcal{X} : \mathcal{X}$ *contains sets* X *of arbitrarily large size*$\}$

Q is *finitely invariant:* if $\mathcal{X}$ and $\mathcal{Y}$ differ by finitely many sets, then $\quad \mathcal{X} \in \mathsf{Q} \Leftrightarrow \mathcal{Y} \in \mathsf{Q}$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow\ (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w,X \models \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X :\ w,X \models \varphi\}$

Q - a property of sets $\mathcal{X}$; $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathsf{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set X*$\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets X of same size*$\}$

$\mathsf{U} = \{\mathcal{X} : \mathcal{X}$ *contains sets X of arbitrarily large size*$\}$

Q is *finitely invariant:* if $\mathcal{X}$ and $\mathcal{Y}$ differ by finitely many sets, then $\quad \mathcal{X} \in \mathsf{Q} \Leftrightarrow \mathcal{Y} \in \mathsf{Q}$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega :\ \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w, X \vDash \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X : w, X \vDash \varphi\}$

Q - a property of sets $\mathcal{X}$; $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathsf{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets* X *of same size*$\}$

$\mathsf{U} = \{\mathcal{X} : \mathcal{X}$ *contains sets* X *of arbitrarily large size*$\}$

Q is *finitely invariant:* if $\mathcal{X}$ and $\mathcal{Y}$ differ by finitely many sets, then $\quad \mathcal{X} \in \mathsf{Q} \Leftrightarrow \mathcal{Y} \in \mathsf{Q}$

Goal: convert a (WMSO+Q)-formula into a boolean combination of Q-formulas, which defines the same language.

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;
  eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w,X \models \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X : w,X \models \varphi\}$

Q - a property of sets $\mathcal{X}$: $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathrm{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets* X *of same size*$\}$

$\mathsf{U} = \{\mathcal{X} : \mathcal{X}$ *contains sets* X *of arbitrarily large size*$\}$

Q is *finitely invariant:* if $\mathcal{X}$ and $\mathcal{Y}$ differ by finitely many sets, then $\quad \mathcal{X} \in \mathsf{Q} \Leftrightarrow \mathcal{Y} \in \mathsf{Q}$

Goal: convert a (WMSO+Q)-formula into a boolean combination of Q-formulas, which defines the same language.

What language does a formula φ with a free variable define?
A language L over $A \times \{0,1\}$:

$$L = \{w \otimes X : \quad w,X \models \varphi \}$$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

eg. $\varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x \in X \Leftrightarrow y \notin X)$

X - a finite set of positions; $\quad w,X \models \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X : w,X \models \varphi\}$

Q - a property of sets $\mathcal{X}$; $\quad \mathsf{Q}X\ \varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\ \varphi(X)$ – a Q-formula

$\exists_{\mathsf{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets* X *of same size*$\}$

$\mathsf{U} = \{\mathcal{X} : \mathcal{X}$ *contains sets* X *of arbitrarily large size*$\}$

Q is *finitely invariant:* if $\mathcal{X}$ and $\mathcal{Y}$ differ by finitely many sets, then $\quad \mathcal{X} \in \mathsf{Q} \Leftrightarrow \mathcal{Y} \in \mathsf{Q}$

Goal: convert a (WMSO+Q)-formula into a boolean combination of Q-formulas, which defines the same language.

What language does a formula φ with a free variable define?
A language L over $A \times \{0,1\}$:

$$L = \{w \otimes X : \quad w,X \models \varphi\}$$

We need to show: *if L is a boolean combination of Q-formulas, then so is*

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L = \{w \in A^\omega : \ \mathsf{Q}X\,[w \otimes X \in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.

φ - a WMSO formula with a free variable X;

$eg.\ \varphi = \forall x\ \forall y \quad \mathrm{suc}(x,y) \Rightarrow (x\in X \Leftrightarrow y\notin X)$

X - a finite set of positions; $\quad w,X \vDash \varphi$

$\mathcal{X}$ - a family of sets X; $\quad \mathcal{X}_\varphi = \{X : w,X\vDash \varphi\}$

Q - a property of sets $\mathcal{X}$; $\quad \mathsf{Q}X\,\varphi(X)$ iff $\mathcal{X}_\varphi \in \mathsf{Q}$

$\mathsf{Q}X\,\varphi(X)$ – a Q-formula

$\exists_{\mathrm{fin}} = \{\mathcal{X} : \mathcal{X}$ *contains some set* $X\}$

$\mathsf{R} = \{\mathcal{X} : \mathcal{X}$ *contains infinitely many sets* X *of same size*$\}$

$\mathsf{U} = \{\mathcal{X} : \mathcal{X}$ *contains sets* X *of arbitrarily large size*$\}$

Q is *finitely invariant:* if $\mathcal{X}$ and $\mathcal{Y}$ differ by finitely many sets, then $\quad \mathcal{X}\in\mathsf{Q} \Leftrightarrow \mathcal{Y}\in\mathsf{Q}$

Goal: convert a (WMSO+Q)-formula into a boolean combination of Q-formulas, which defines the same language.

What language does a formula φ with a free variable define?

A language L over $A \times \{0,1\}$:

$$L=\{w\otimes X: \quad w,X \vDash \varphi\}$$

We need to show: *if L is a boolean combination of Q-formulas, then so is*

$$\mathsf{Q}\,L=\{w \in A^\omega: \ \mathsf{Q}X\,[w\otimes X\in L]\}$$

Theorem. Let F be a prefix-independent acceptance condition and let Q be a locus quantifier. If L is an F-regular language over the alphabet $A \times \{0,1\}$, then the language

$$\mathsf{Q}\,L=\{w \in A^\omega: \ \mathsf{Q}X\,[w\otimes X\in L]\}$$

is a boolean combination of F-regular languages and Q-formulas. Moreover, if Q is prefix-independent then the Q-formulas are prefix Q-formulas.