On the complexity of regular languages

Corentin Barloy

Supervised by: Charles Paperman, Michaël Cadilhac and Sylvain Salvati

Université de Lille

Optimisation of sequential computations

Sequential vs Parallel

Does the string contain aa:

b	a	b	a	a	b

Sequential vs Parallel

Does the string contain aa:

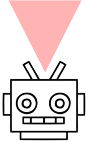
b	a	b	a	a	b

Sequential: one operation at a time

Sequential vs Parallel

Does the string contain *aa*:

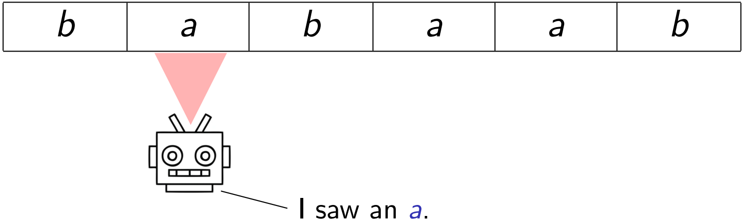
b	*a*	*b*	*a*	*a*	*b*



Sequential: one operation at a time

Sequential vs Parallel

Does the string contain *aa*:

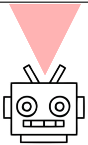


Sequential: one operation at a time

Sequential vs Parallel

Does the string contain *aa*:

b	*a*	*b*	*a*	*a*	*b*

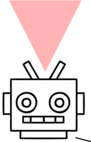


Sequential: one operation at a time

Sequential vs Parallel

Does the string contain aa:

b	a	b	a	a	b

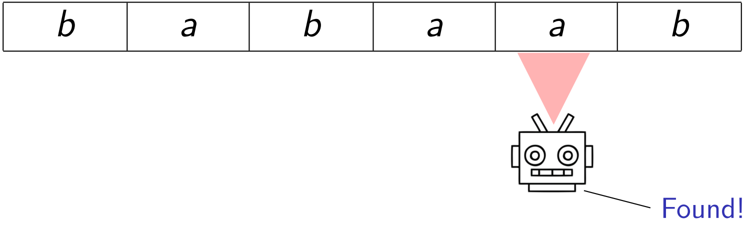


I saw an a.

Sequential: one operation at a time

Sequential vs Parallel

Does the string contain *aa*:

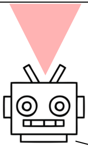


Sequential: one operation at a time

Sequential vs Parallel

Does the string contain *aa*:

b	*a*	*b*	*a*	*a*	*b*



Found!

Sequential: one operation at a time

Sequential vs Parallel

Does the string contain aa:

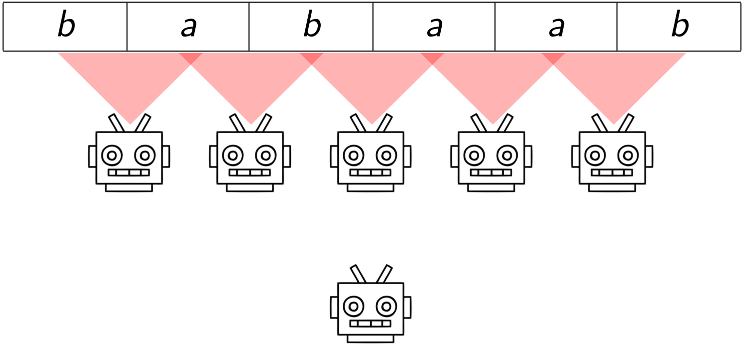
b	a	b	a	a	b

Sequential: one operation at a time

Parallel: several operations at the same time

Sequential vs Parallel

Does the string contain *aa*:

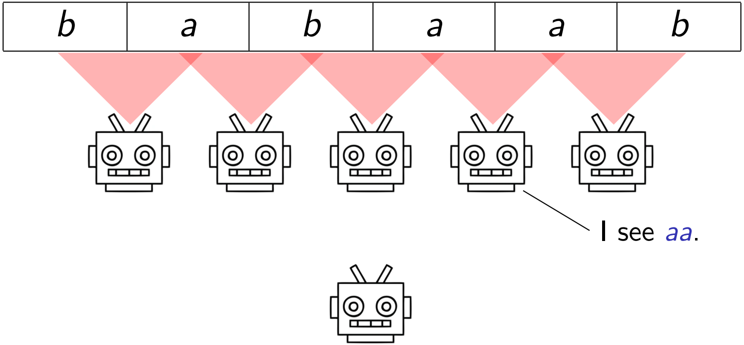


Sequential: one operation at a time

Parallel: several operations at the same time

Sequential vs Parallel

Does the string contain *aa*:



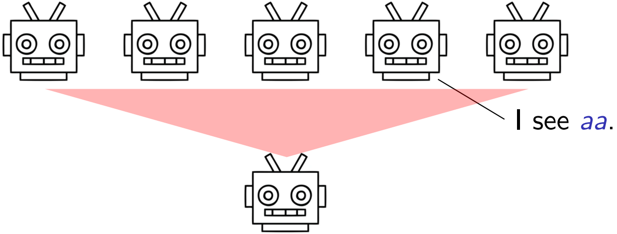
Sequential: one operation at a time

Parallel: several operations at the same time

Sequential vs Parallel

Does the string contain aa:

b	a	b	a	a	b



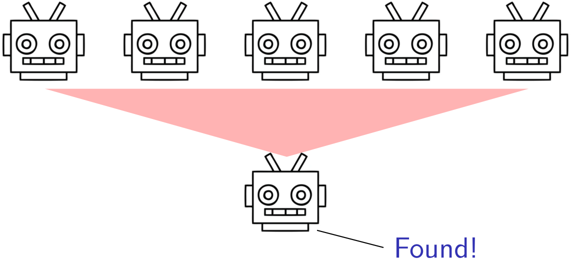
Sequential: one operation at a time

Parallel: several operations at the same time

Sequential vs Parallel

Does the string contain *aa*:

b	*a*	*b*	*a*	*a*	*b*



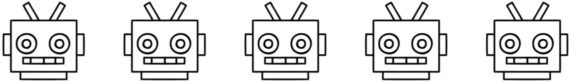
Sequential: one operation at a time

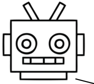
Parallel: several operations at the same time

Sequential vs Parallel

Does the string contain aa:

b	a	b	a	a	b





Found!

Sequential: one operation at a time $\rightarrow$ Finite Automata

Parallel: several operations at the same time $\rightarrow$ Boolean Circuit

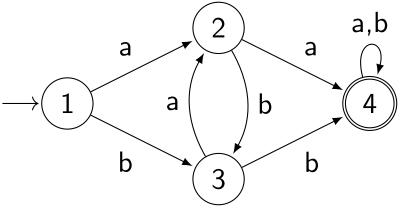
Finite automata

Finite automata

A simple sequential model:

Finite automata

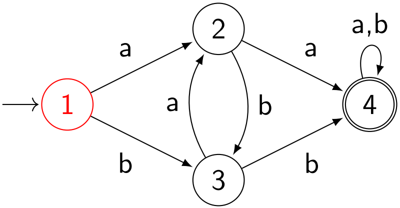
A simple sequential model:



Finite automata

A simple sequential model:

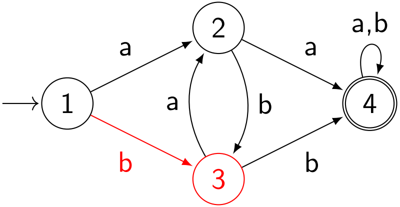
w=babaab



Finite automata

A simple sequential model:

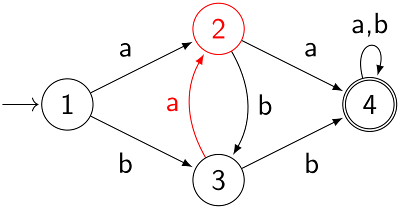
w=babaab



Finite automata

A simple sequential model:

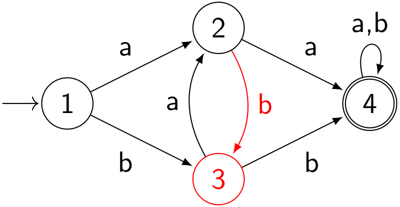
w=babaab



Finite automata

A simple sequential model:

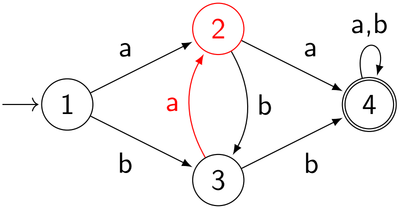
w=babaab



Finite automata

A simple sequential model:

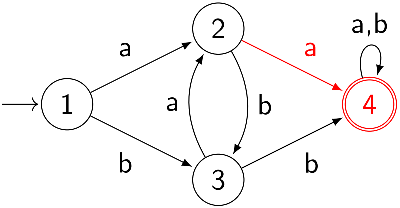
w=babaab



Finite automata

A simple sequential model:

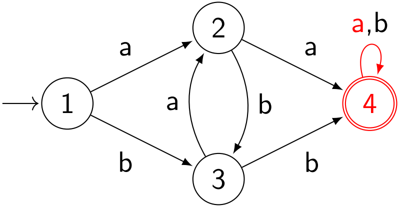
w=babaab



Finite automata

A simple sequential model:

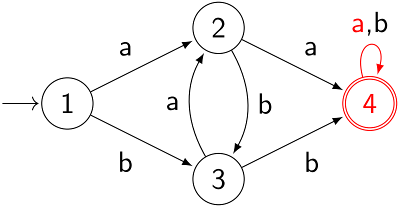
w=babaab



Finite automata

A simple sequential model:

w=babaab

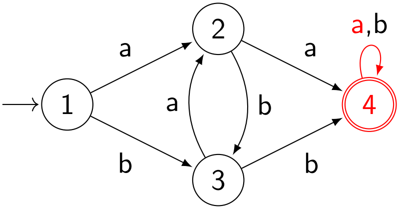


Computes Twice = contains *aa* or *bb*

Finite automata

A simple sequential model:

w=babaab



Computes Twice = contains *aa* or *bb*

Ubiquitous:

- Data extraction
- Compilers
- Text editors
- Linguistics
- Bioinformatic
- Security
- ...

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties

Algebra

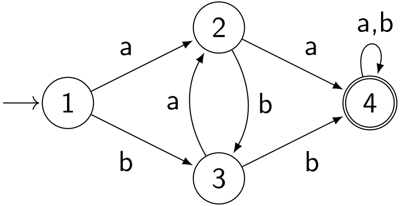
Many classes of regular languages enjoy several closure properties: they form varieties

$\rightarrow$ their languages can be understood algebraically

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



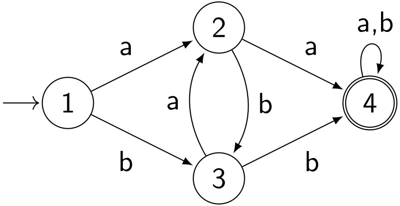
Syntactic monoid

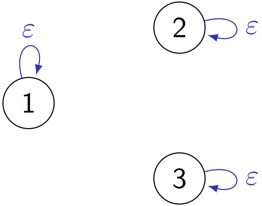
	1	2	3	4

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton





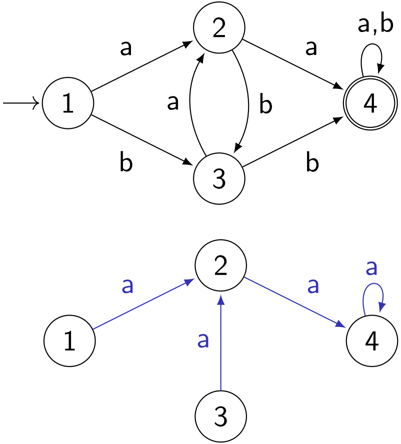
Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



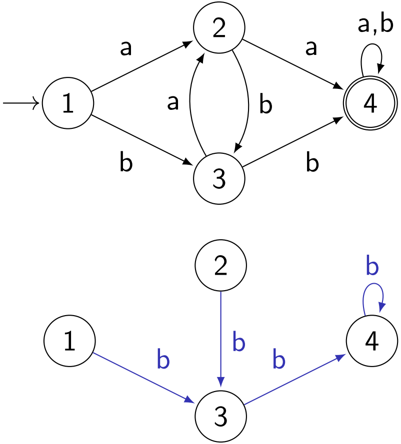
Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



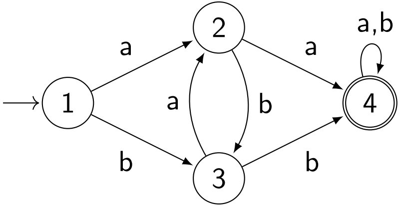
Syntactic monoid

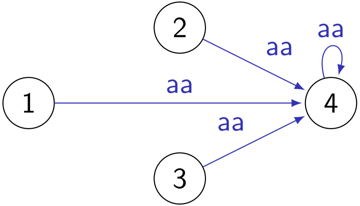
	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
→ their languages can be understood algebraically

Minimal automaton





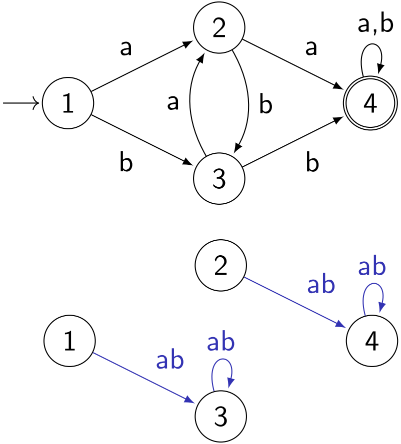
Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



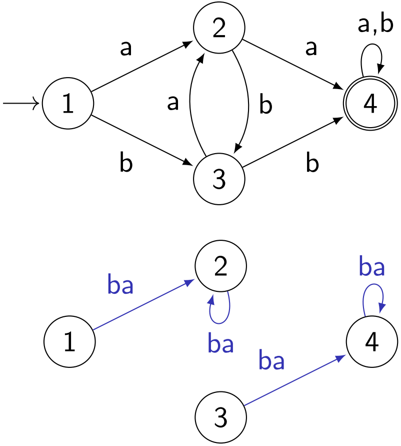
Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
→ their languages can be understood algebraically

Minimal automaton



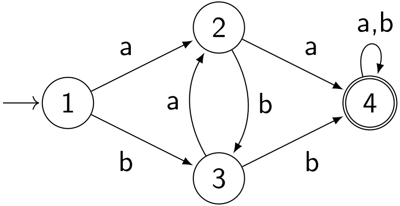
Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

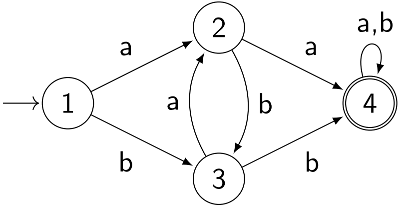
	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

$$\begin{aligned} f_{bb} &= f_{aa} \\ f_{aba} &= f_a \\ f_{bab} &= f_b \end{aligned}$$

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

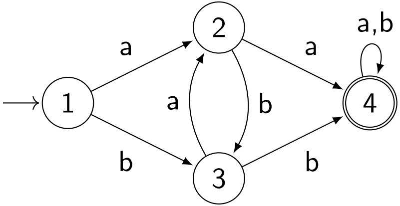
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

f_{ababab}

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

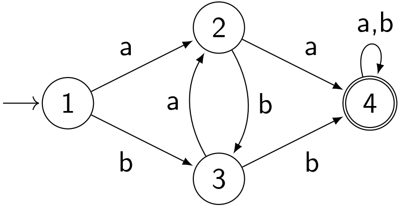
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

$$f_a \circ f_{bab} \circ f_{ab}$$

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

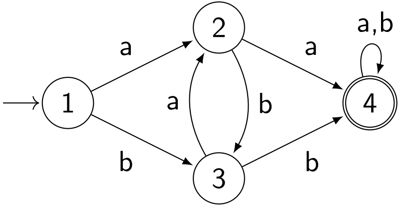
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

$$f_a \circ f_b \circ f_{ab}$$

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

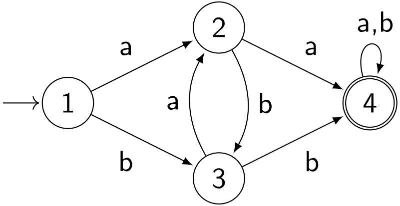
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

$$f_{abab}$$

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

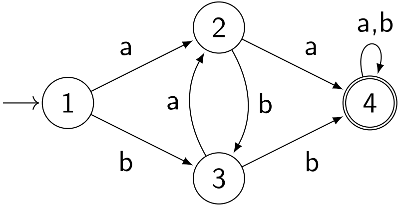
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

$$f_{aba} \circ f_b$$

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

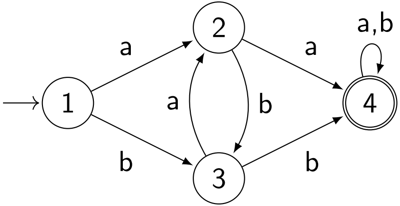
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

$$f_a \circ f_b$$

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

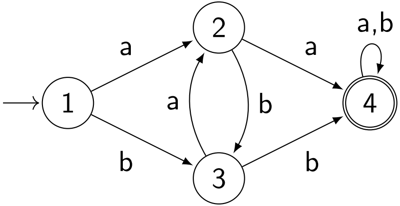
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

f_{ab}

Algebra

Many classes of regular languages enjoy several closure properties: they form varieties
$\rightarrow$ their languages can be understood algebraically

Minimal automaton



Syntactic monoid

	1	2	3	4
f_ε	1	2	3	4
f_a	2	4	2	4
f_b	3	3	4	4
f_{aa}	4	4	4	4
f_{ab}	3	4	3	4
f_{ba}	2	2	4	4

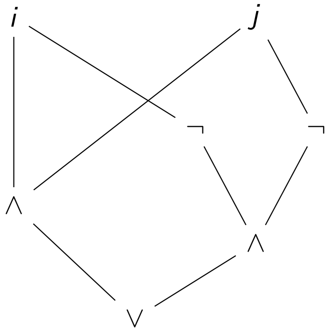
$$f_{bb} = f_{aa}$$
$$f_{aba} = f_a$$
$$f_{bab} = f_b$$

f_{ab}

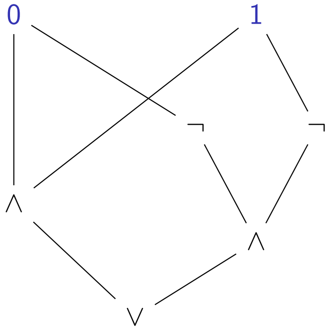
We can look at algebraic properties instead of combinatorial ones
Ex: idempotents

Boolean circuits

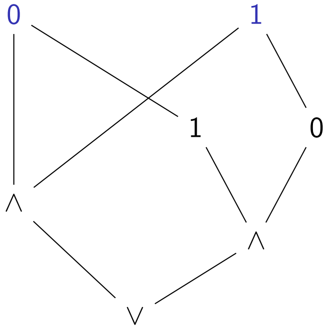
Circuits



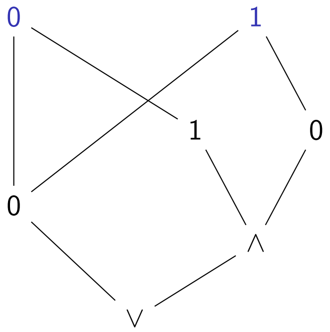
Circuits



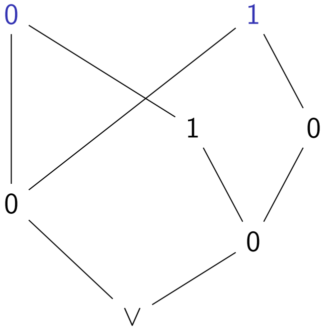
Circuits



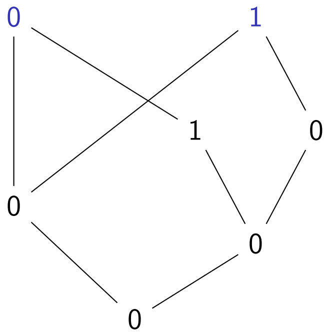
Circuits



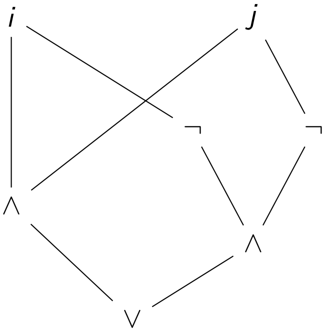
Circuits



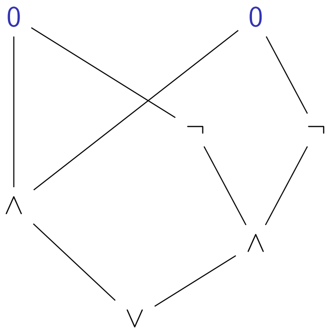
Circuits



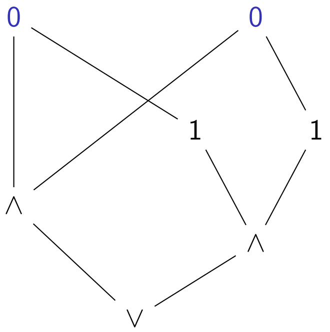
Circuits



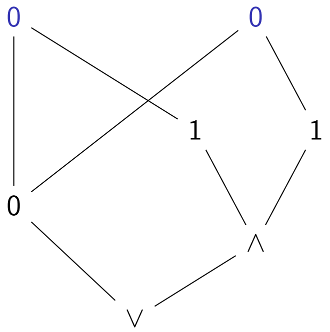
Circuits



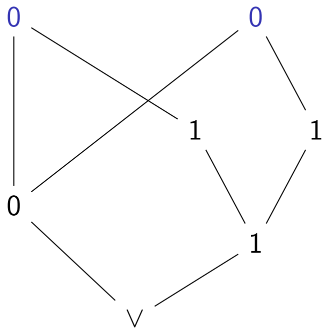
Circuits



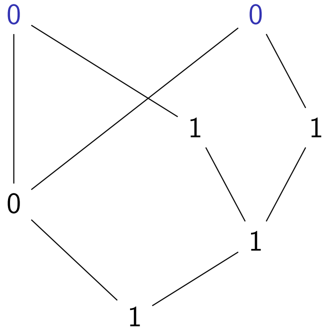
Circuits



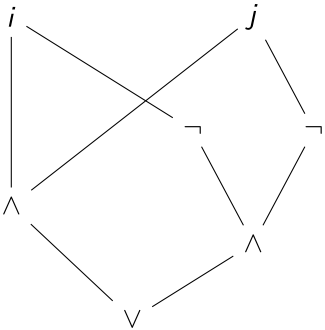
Circuits



Circuits

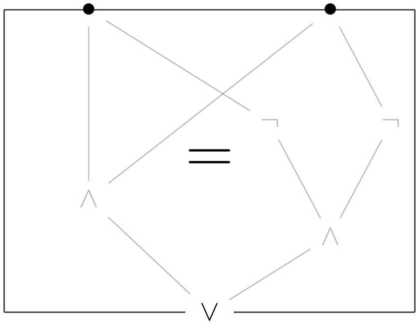


Circuits



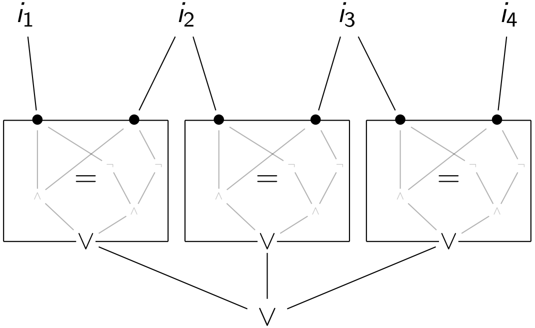
outputs $1 \Leftrightarrow i = j$

Circuits

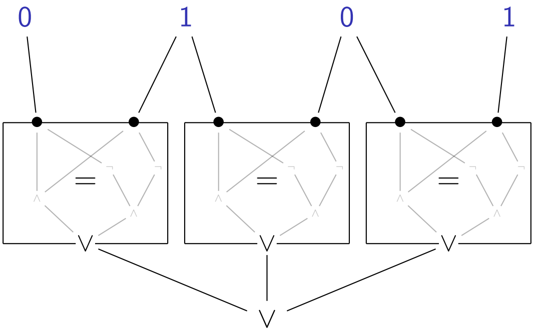


outputs $1 \Leftrightarrow i = j$

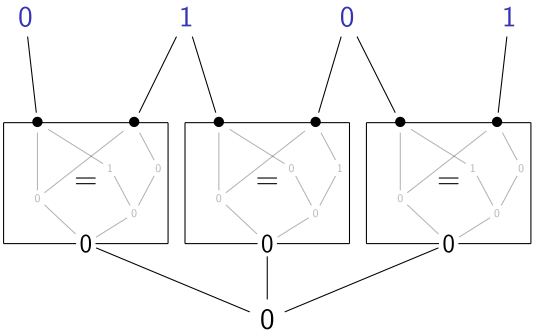
Circuits



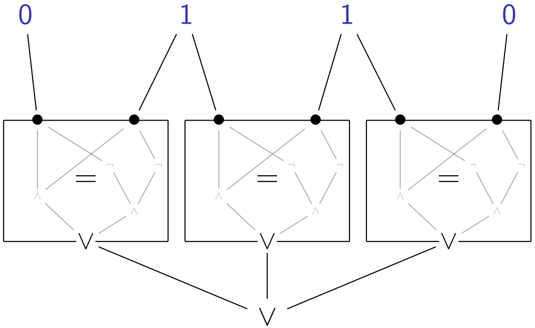
Circuits



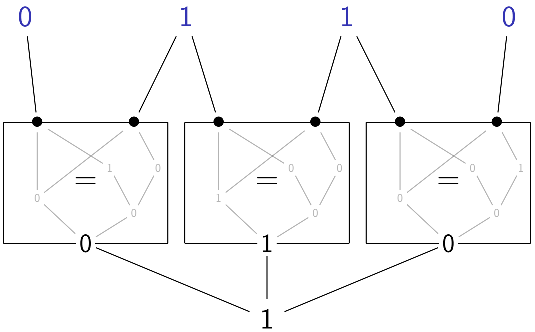
Circuits



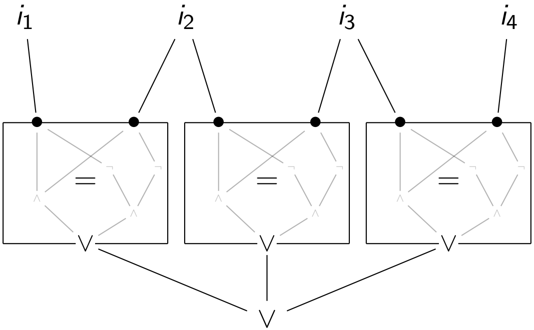
Circuits



Circuits

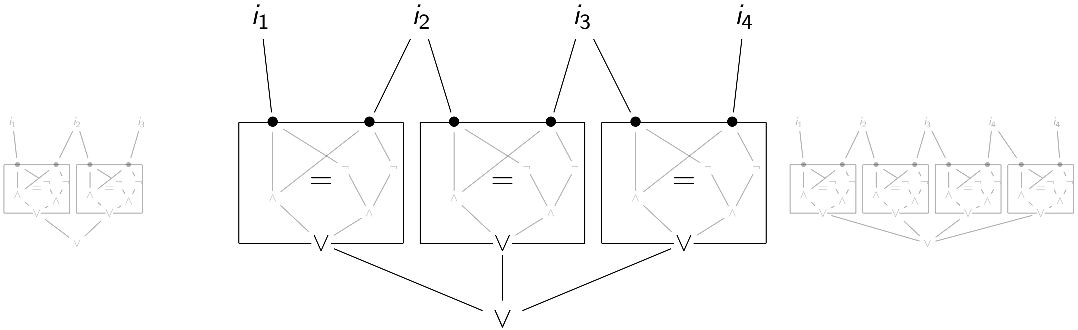


Circuits



Computes Twice

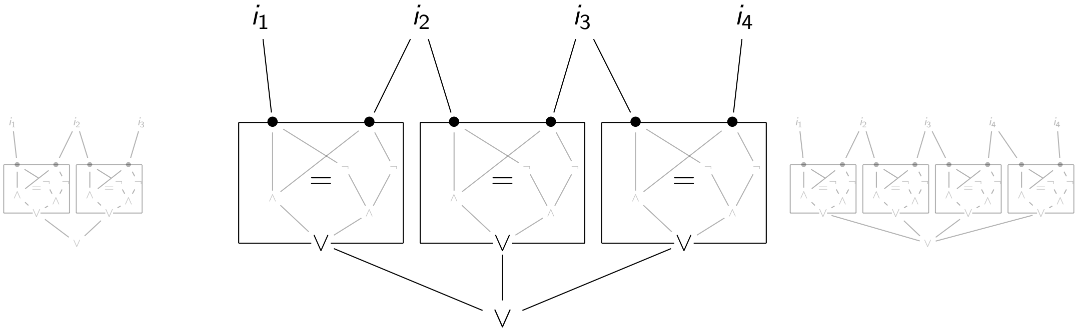
Circuits



Computes Twice

Circuits

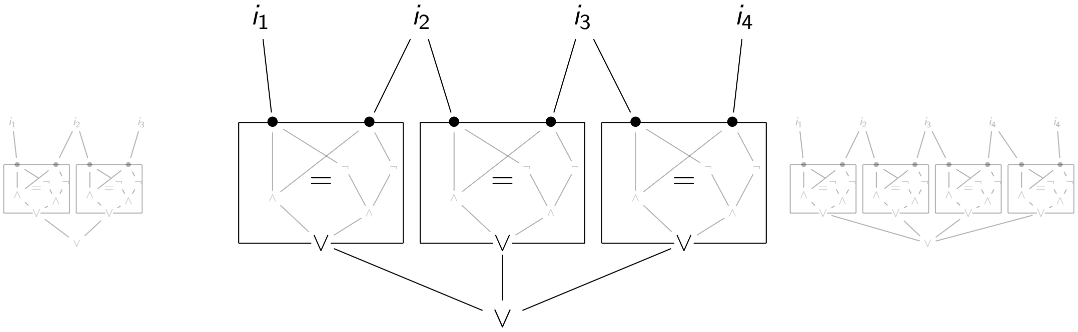
$$\text{size}(n) = \text{number of gates}$$
$$\text{depth}(n) = \text{maximal length of a path}$$
$$\text{fan-in}(n) = \text{maximal in-degree of a gate}$$



Computes Twice

Circuits

$$\text{size}(n) = \text{number of gates}$$
$$\text{depth}(n) = \text{maximal length of a path}$$
$$\text{fan-in}(n) = \text{maximal in-degree of a gate}$$



Computes Twice

$\text{size}(n) = 5 \cdot (n-1) + 1$ $\text{depth}(n) = 4$ $\text{fan-in}(n) = n - 1$

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

$\rightarrow$ Study regular languages in circuit classes

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

$\rightarrow$ Study regular languages in circuit classes

poly size
constant depth
unbounded fan-in
$+$ mod counting

poly size
constant depth
unbounded fan-in

poly size
log depth
bounded fan-in

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

$\rightarrow$ Study regular languages in circuit classes

Even $\#$1s
Furst, Saxe, Sipser (1984)

poly size
constant depth
unbounded fan-in
$+$ mod counting

poly size
constant depth
unbounded fan-in

poly size
log depth
bounded fan-in

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

$\rightarrow$ Study regular languages in circuit classes

Even #1s
Furst, Saxe, Sipser (1984)

poly size
constant depth
unbounded fan-in
$+$ mod counting

same languages
iff
same regular languages
Barrington (1992)

$\subsetneq$

$\subseteq$

poly size
constant depth
unbounded fan-in

poly size
log depth
bounded fan-in

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

$\rightarrow$ Study regular languages in circuit classes

Even #1s
Furst, Saxe, Sipser (1984)

same languages
iff
same regular languages
Barrington (1992)

poly size constant depth unbounded fan-in	$\subsetneq$	poly size constant depth unbounded fan-in + mod counting	$\subseteq$	poly size log depth bounded fan-in

$\subseteq$

NL

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

$\rightarrow$ Study regular languages in circuit classes

Even $\#$1s
Furst, Saxe, Sipser (1984)

poly size
constant depth
unbounded fan-in

$\subsetneq$

poly size
constant depth
unbounded fan-in
$+$ mod counting

$\not\supseteq$

NEXP

same languages
iff
same regular languages
Barrington (1992)

$\subseteq$

poly size
log depth
bounded fan-in

$\subseteq$

NL

Regular languages in circuit classes

Which regular languages can be parallelized efficiently?

$\rightarrow$ Study regular languages in circuit classes

Even #1s
Furst, Saxe, Sipser (1984)

poly size
constant depth
unbounded fan-in

$\subsetneq$

poly size
constant depth
unbounded fan-in
+ mod counting

$\not\supseteq$

NEXP

same languages
iff
same regular languages
Barrington (1992)

$\subseteq$

poly size
log depth
bounded fan-in

$\subseteq$

NL

Many more (bounded depth classes, addition, ...)

Logic

x, y two positions in a string:

$$\mathrm{eq}(x, y) = (a(x) \wedge a(y)) \vee (b(x) \wedge b(y))$$

Logic

x, y two positions in a string:

Letter predicate

$$\mathrm{eq}(x, y) = (a(x) \wedge a(y)) \vee (b(x) \wedge b(y))$$

Logic

x, y two positions in a string:

Letter predicate

$$\mathrm{eq}(x, y) = (a(x) \wedge a(y)) \vee (b(x) \wedge b(y))$$

Formulas are interpreted over strings:

$$\exists x \; \exists y \; (x = y + 1) \wedge \mathrm{eq}(x, y)$$

Logic

x, y two positions in a string:

Letter predicate

$$\mathrm{eq}(x,y) = (a(x) \wedge a(y)) \vee (b(x) \wedge b(y))$$

Formulas are interpreted over strings:

Numerical predicate

$$\exists x\ \exists y\ (x = y + 1) \wedge \mathrm{eq}(x,y)$$

Quantification over a position

Logic

x, y two positions in a string:

Letter predicate

$$\mathrm{eq}(x, y) = (a(x) \wedge a(y)) \vee (b(x) \wedge b(y))$$

Formulas are interpreted over strings:

Numerical predicate

$$\exists x\ \exists y\ (x = y + 1) \wedge \mathrm{eq}(x, y)$$

Quantification over a position

b	a	b	a	a	b

Logic

x, y two positions in a string:

Letter predicate

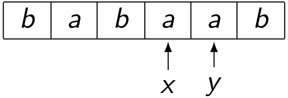
$$\mathrm{eq}(x,y) = (a(x) \wedge a(y)) \vee (b(x) \wedge b(y))$$

Formulas are interpreted over strings:

Numerical predicate

$$\exists x\ \exists y\ (x = y + 1) \wedge \mathrm{eq}(x,y)$$

Quantification over a position



Logic

x, y two positions in a string:

Letter predicate

$$\mathrm{eq}(x,y) = (a(x) \wedge a(y)) \vee (b(x) \wedge b(y))$$

Formulas are interpreted over strings:

Numerical predicate

$$\exists x\ \exists y\ (x = y + 1) \wedge \mathrm{eq}(x,y)$$

Quantification over a position

b	a	b	a	a	b
			$\uparrow$ x	$\uparrow$ y	

Describes Twice

Logic

Fragments

Numerical predicates

Logic

Fragments

- First-order logic: FO

Numerical predicates

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k

Numerical predicates

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

- The order: $x < y$

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

- The order: $x < y$
- The successor predicate: $x + 1 = y$

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

- The order: $x < y$
- The successor predicate: $x + 1 = y$
- The modular predicates: $x \bmod 3 = 0$

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

- The order: $x < y$
- The successor predicate: $x + 1 = y$
- The modular predicates: $x \bmod 3 = 0$

Regular predicates: REG

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

- The order: $x < y$
- The successor predicate: $x + 1 = y$
- The modular predicates: $x \bmod 3 = 0$

Regular predicates: REG

- Many more ($xy = z$, encoding of a cat picture, ...)

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

- The order: $x < y$
- The successor predicate: $x + 1 = y$
- The modular predicates: $x \bmod 3 = 0$

Regular predicates: REG

- Many more ($xy = z$, encoding of a cat picture, ...)

Arbitrary predicates: ARB

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

Numerical predicates

- The order: $x < y$
- The successor predicate: $x + 1 = y$
- The modular predicates: $x \bmod 3 = 0$

Regular predicates: REG

- Many more ($xy = z$, encoding of a cat picture, ...)

Arbitrary predicates: ARB

$$\exists x \exists y\ (x = y + 1) \wedge \mathrm{eq}(x, y) \in \Sigma_1[+1]$$

Logic

Fragments

- First-order logic: FO
- Bounded quantifier alternation: Σ_k, Π_k
- Many more (restricting the number of variables, adding modular quantifiers,...)

$$\exists x \exists y \ (x = y + 1) \wedge \mathrm{eq}(x, y)$$
$$\in \Sigma_1[+1]$$

Numerical predicates

- The order: $x < y$
- The successor predicate: $x + 1 = y$
- The modular predicates: $x \bmod 3 = 0$

Regular predicates: REG

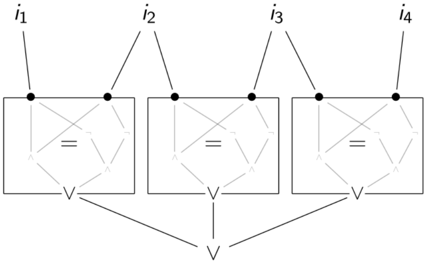
- Many more ($xy = z$, encoding of a cat picture, ...)

Arbitrary predicates: ARB

$\forall x,$  $(x) \Rightarrow a(x)$

$\in \Pi_1[\mathrm{ARB}]$

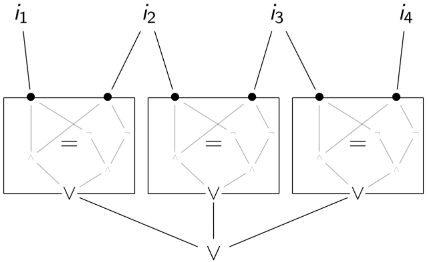
Logic and circuits



$\longleftrightarrow$

$$\exists x \exists y \ (x = y + 1) \wedge \mathrm{eq}(x, y)$$

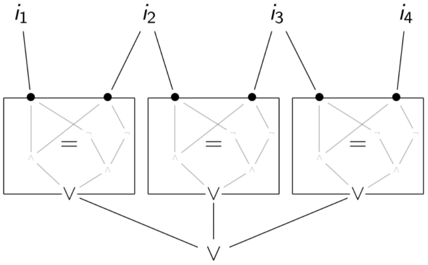
Logic and circuits



$\longleftrightarrow$ $\exists x \exists y \ (x = y + 1) \wedge \mathrm{eq}(x, y)$

Constant depth circuits $\longleftrightarrow$ FO[ARB]

Logic and circuits



$\longleftrightarrow$ $\exists x \exists y\ (x = y + 1) \wedge \mathrm{eq}(x, y)$

Constant depth circuits $\longleftrightarrow$ FO[ARB]

depth $\longleftrightarrow$ quantifiers alternation

Straubing's properties

For $\mathcal{L}$ a logical fragment,

What are the regular languages of $\mathcal{L}[ARB]$ *?*

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

A natural guess (Straubing):

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \quad !$$

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \ !$$

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \ \ !$$

True

- Σ_1
- FO
- FO with prime modular quantifiers

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \quad !$$

True

- Σ_1
- FO $\longleftrightarrow$ constant depth
- FO with prime modular quantifiers

constant depth + mod counting

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$ *?*

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \;\; !$$

True	False
▶ Σ_1	▶ FO + S_5
▶ FO ←→ constant depth	▶ FO ∘ MOD
▶ FO with prime modular quantifiers	

constant depth
+ mod counting

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \ !$$

True	False	Open
► Σ_1 ► FO ←→ constant depth ► FO with prime modular quantifiers	► FO + S_5 ► FO ∘ MOD	► FO with composite modular quantifiers ► FO with two variables ► Σ_k, $k \geq 3$

constant depth
+ mod counting

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$ *?*

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \;\; !$$

Circuits with
composite modular gates
(AND with MOD_6 gates)

True	False	Open
▶ Σ_1 ▶ FO ⟷ constant depth ▶ FO with prime modular quantifiers	▶ $\mathrm{FO} + S_5$ ▶ $\mathrm{FO} \circ \mathrm{MOD}$	▶ FO with composite modular quantifiers ▶ FO with two variables ▶ Σ_k, $k \geq 3$

constant depth
+ mod counting

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \ !$$

Circuits with composite modular gates (AND with MOD_6 gates)

True	False	Open
▶ Σ_1 ▶ FO ⟷ constant depth ▶ FO with prime modular quantifiers	▶ $\mathrm{FO} + S_5$ ▶ $\mathrm{FO} \circ \mathrm{MOD}$	▶ FO with composite modular quantifiers ▶ FO with two variables ▶ Σ_k, $k \geq 3$

constant depth + mod counting

Linear size circuits (complexity of addition)

Straubing's properties

For $\mathcal{L}$ a logical fragment,

Are they decidable?

What are the regular languages of $\mathcal{L}[ARB]$?

A natural guess (Straubing):

Usually decidable

$$\mathcal{L}[\mathrm{ARB}] \cap \mathrm{Reg} = \mathcal{L}[\mathrm{REG}] \ !$$

Circuits with composite modular gates (AND with MOD_6 gates)

True	False	Open
Σ_1	FO + S_5	FO with composite modular quantifiers
FO ⟷ constant depth	FO ∘ MOD	FO with two variables
FO with prime modular quantifiers		Σ_k, $k \geq 3$

constant depth + mod counting

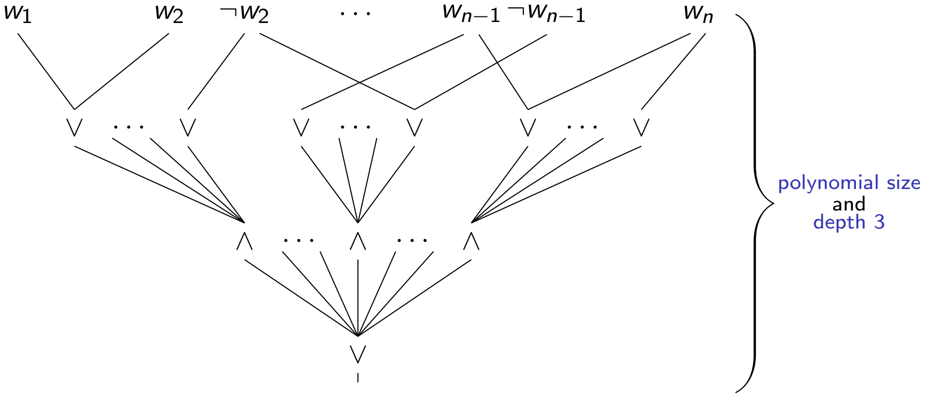
Chap 5: $\Sigma_2[\mathrm{ARB}] \cap \mathrm{Reg} = \Sigma_2[\mathrm{REG}]$

Linear size circuits (complexity of addition)

Straubing property for Σ_2

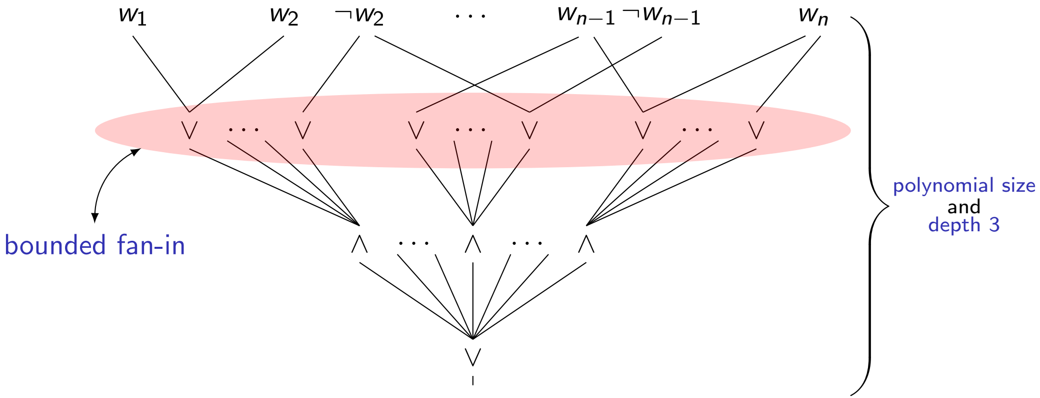
The circuit class Σ_2

$w = w_1 w_2 \cdots w_{n-1} w_n$



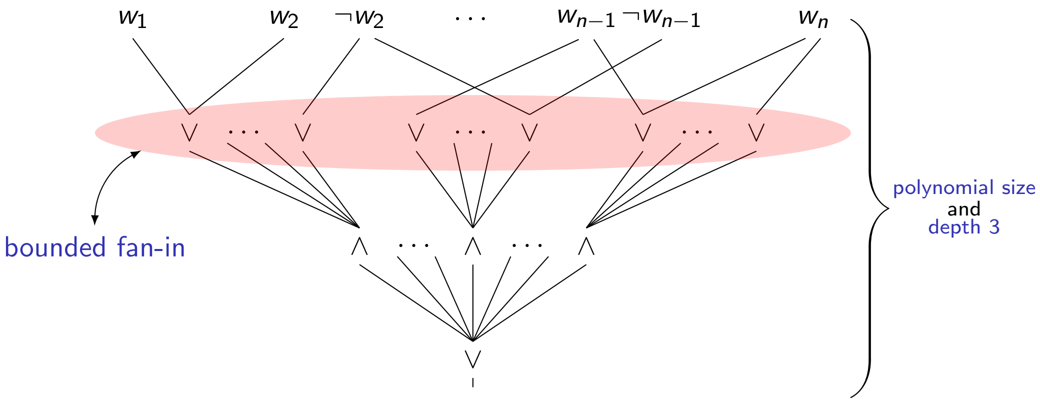
The circuit class Σ_2

$$w = w_1 w_2 \cdots w_{n-1} w_n$$



The circuit class Σ_2

$w = w_1 w_2 \cdots w_{n-1} w_n$



It is equivalent to Σ_2[ARB].

Theorem 5.34: (*Barloy*, Cadilhac, Paperman, Zeume)

$$\Sigma_2[\mathrm{ARB}] \cap \mathrm{Reg} = \Sigma_2[\mathrm{REG}]$$

Theorem 5.34: (*Barloy*, Cadilhac, Paperman, Zeume)

$$\Sigma_2[\mathsf{ARB}] \cap \mathsf{Reg} = \Sigma_2[\mathsf{REG}]$$

- $\supseteq$: Immediate

Theorem 5.34: (*Barloy*, Cadilhac, Paperman, Zeume)

$$\Sigma_2[\mathrm{ARB}] \cap \mathrm{Reg} = \Sigma_2[\mathrm{REG}]$$

- $\supseteq$: Immediate
- $\subseteq$: Take a regular language not in $\Sigma_2[\mathrm{REG}]$, show that it is not in $\Sigma_2[\mathrm{ARB}]$

Theorem 5.34: (*Barloy*, Cadilhac, Paperman, Zeume)

$$\Sigma_2[\mathrm{ARB}] \cap \mathrm{Reg} = \Sigma_2[\mathrm{REG}]$$

- $\supseteq$: Immediate
- $\subseteq$: Take $\underbrace{\text{a regular language not in } \Sigma_2[\mathrm{REG}]}_{\text{Algebra}}$, show that it is not in $\Sigma_2[\mathrm{ARB}]$

Theorem 5.34: (*Barloy*, Cadilhac, Paperman, Zeume)

$$\Sigma_2[\mathsf{ARB}] \cap \mathrm{Reg} = \Sigma_2[\mathsf{REG}]$$

- $\supseteq$: Immediate
- $\subseteq$: Take $\underbrace{\text{a regular language not in } \Sigma_2[\mathsf{REG}]}_{\text{Algebra}}$, show that $\underbrace{\text{it is not in } \Sigma_2[\mathsf{ARB}]}_{\text{Circuit lower bound}}$

Proof sketch

Algebra | Lower bound

Proof sketch

Theorem (Pin, Weil)

$\mathcal{L}$ a language and M its syntactic monoid

Proof sketch

Theorem (Pin, Weil)

$\mathcal{L}$ a language and M its syntactic monoid

$$\mathcal{L} \in \Sigma_2[\mathrm{REG}]$$

iff

$$\forall x, y, z \in M,\ (xyz)^{|M|!} = (xyz)^{|M|!} y (xyz)^{|M|!}$$

Proof sketch

Theorem (Pin, Weil)

$\mathcal{L}$ a language and M its syntactic monoid

$$\mathcal{L} \in \Sigma_2[\mathrm{REG}]$$

iff

$$\forall x, y, z \in M,\ (xyz)^{|M|!} = (xyz)^{|M|!} y (xyz)^{|M|!}$$

Decidable!

Proof sketch

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

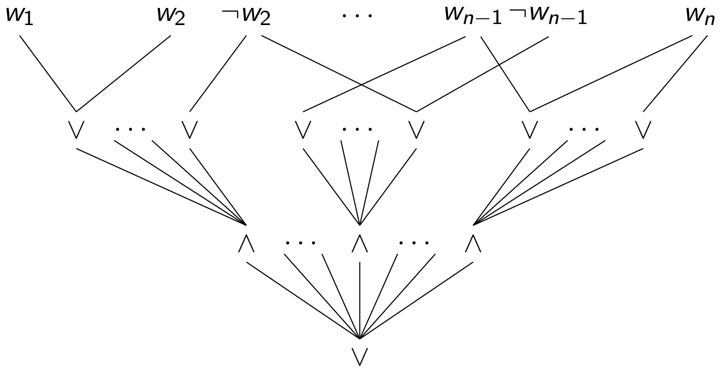
Then C accepts a word outside of $\mathcal{L}$

Proof sketch

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$

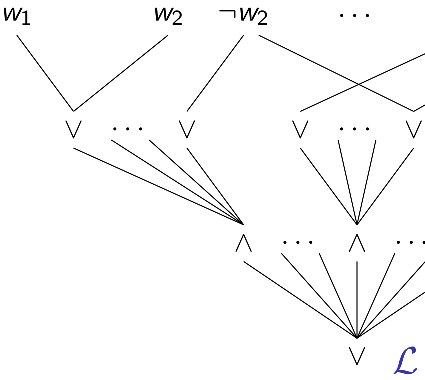


Proof sketch

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$

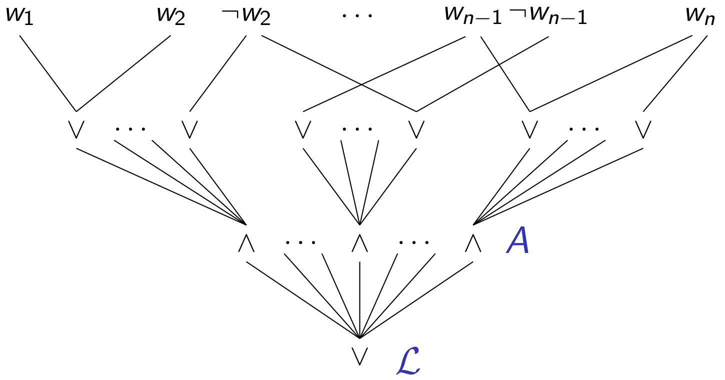


Proof sketch

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$



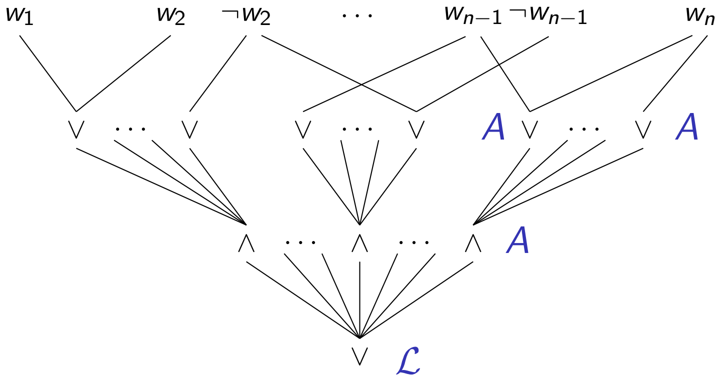
Proof sketch

Algebra | Lower bound

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$



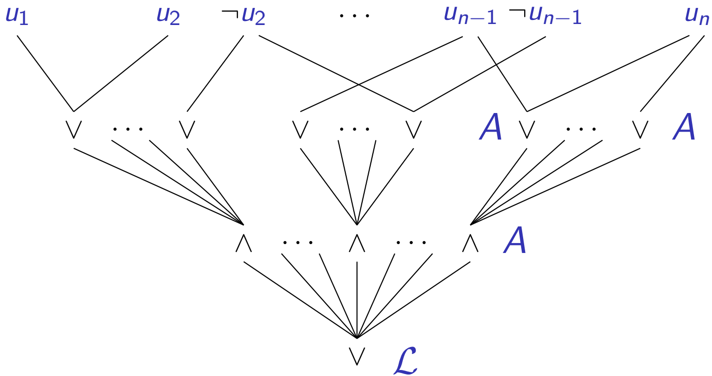
Proof sketch

Algebra | Lower bound

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$



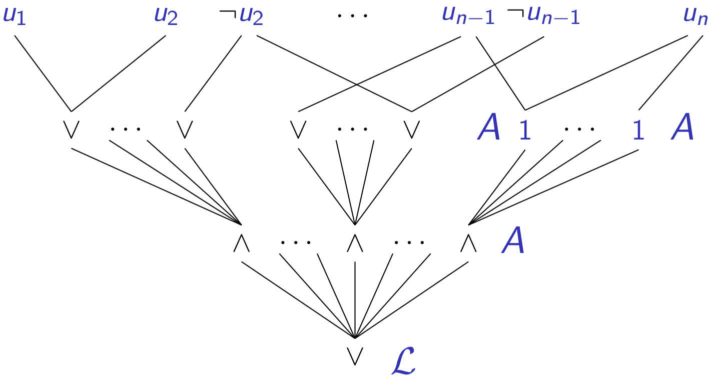
Proof sketch

Algebra | Lower bound

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$

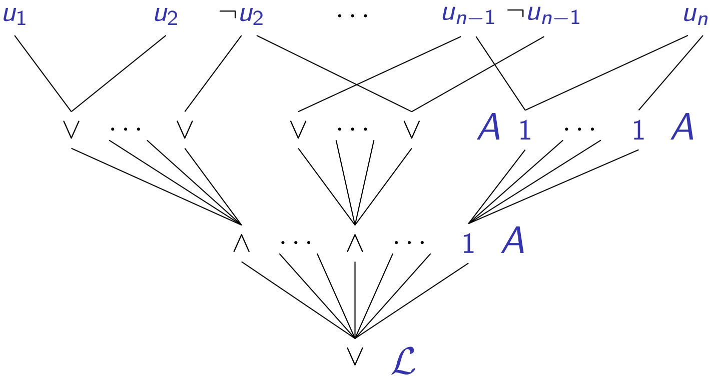


Proof sketch

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$



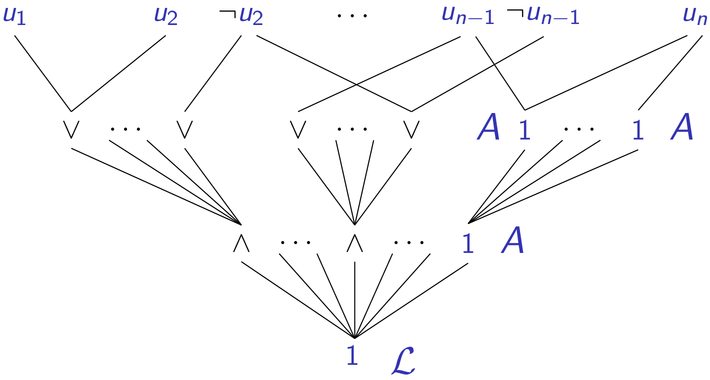
Proof sketch

Algebra | Lower bound

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$



Proof sketch

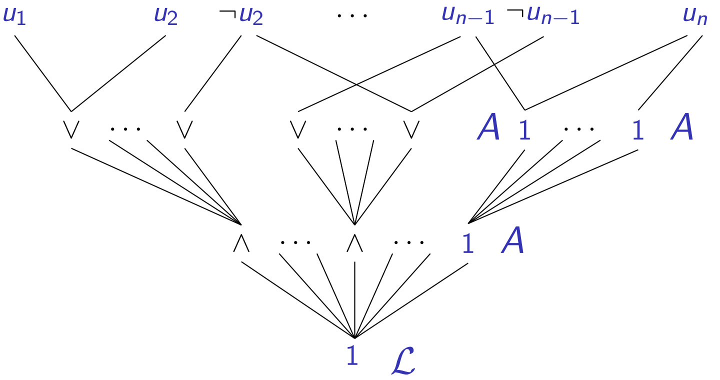
Algebra | Lower bound

Prop: (Håstad, Jukna, Pudlák)
With C a Σ_2 circuit for $\mathcal{L}$, top fan-in k
If for all large $A \subseteq \mathcal{L}$ there exists $u \notin \mathcal{L}$ s.t.

Find it!

For any k pos of u, there is a string in A that matches u on these

Then C accepts a word outside of $\mathcal{L}$



Other optimisations

Other optimisations

XML complexity: Process regular properties of large serialized trees

Other optimisations

XML complexity: Process regular properties of large serialized trees

$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

Other optimisations

XML complexity: Process regular properties of large serialized trees

→ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

→ speed-up JSON parsing: RSONPath

Other optimisations

XML complexity: Process regular properties of large serialized trees

$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

$\rightarrow$ speed-up JSON parsing: RSONPath

Coauthors

Chap 6:
Murlak
Paperman

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees

$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

$\rightarrow$ speed-up JSON parsing: RSONPath

Chap 6:
Murlak
Paperman

Incremental complexity: the input changes over time

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees
$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

$\rightarrow$ speed-up JSON parsing: RSONPath

Chap 6:
Murlak
Paperman

Incremental complexity: the input changes over time
$\rightarrow$ maintain auxiliary data in the RAM model

Other optimisations

XML complexity: Process regular properties of large serialized trees
$\rightarrow$ space trichotomy for path languages:
$O(1)$ $\Theta(\log(n))$ $\Theta(n)$
$\rightarrow$ speed-up JSON parsing: RSONPath

Incremental complexity: the input changes over time
$\rightarrow$ maintain auxiliary data in the RAM model
$\rightarrow$ Already known: trichotomy for update time:
$O(1)$ $\Theta(\log(\log(n)))$ $\Theta(\log(n))$

Coauthors

Chap 6:
Murlak
Paperman

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees

$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

$\rightarrow$ speed-up JSON parsing: RSONPath

Chap 6: Murlak Paperman

Incremental complexity: the input changes over time

$\rightarrow$ maintain auxiliary data in the RAM model

$\rightarrow$ Already known: trichotomy for update time:

$O(1)$ $\Theta(\log(\log(n)))$ $\Theta(\log(n))$

$\rightarrow$ Partial extension to tree languages

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees

$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

$\rightarrow$ speed-up JSON parsing: RSONPath

Chap 6:
Murlak
Paperman

Incremental complexity: the input changes over time

$\rightarrow$ maintain auxiliary data in the RAM model

$\rightarrow$ Already known: trichotomy for update time:

$O(1)$ $\Theta(\log(\log(n)))$ $\Theta(\log(n))$

$\rightarrow$ Partial extension to tree languages

Chap 7:
Amarilli
Jachiet
Paperman

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees
→ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

→ speed-up JSON parsing: RSONPath

Chap 6:
Murlak
Paperman

Incremental complexity: the input changes over time
→ maintain auxiliary data in the RAM model
→ Already known: trichotomy for update time:

$O(1)$ $\Theta(\log(\log(n)))$ $\Theta(\log(n))$

→ Partial extension to tree languages

Chap 7:
Amarilli
Jachiet
Paperman

→ maintain auxiliary data in relational databases

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees
$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

$\rightarrow$ speed-up JSON parsing: RSONPath

Chap 6:
Murlak
Paperman

Incremental complexity: the input changes over time
$\rightarrow$ maintain auxiliary data in the RAM model
$\rightarrow$ Already known: trichotomy for update time:

$O(1)$ $\Theta(\log(\log(n)))$ $\Theta(\log(n))$

$\rightarrow$ Partial extension to tree languages

Chap 7:
Amarilli
Jachiet
Paperman

$\rightarrow$ maintain auxiliary data in relational databases
$\rightarrow$ Updates with FO formulas

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees
$\rightarrow$ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

$\rightarrow$ speed-up JSON parsing: RSONPath

Chap 6:
Murlak
Paperman

Incremental complexity: the input changes over time
$\rightarrow$ maintain auxiliary data in the RAM model
$\rightarrow$ Already known: trichotomy for update time:

$O(1)$ $\Theta(\log(\log(n)))$ $\Theta(\log(n))$

$\rightarrow$ Partial extension to tree languages

Chap 7:
Amarilli
Jachiet
Paperman

$\rightarrow$ maintain auxiliary data in relational databases
$\rightarrow$ Updates with FO formulas
$\rightarrow$ Classification of regular languages depending on:
- arity of relations
- complexity of formulas

Other optimisations

Coauthors

XML complexity: Process regular properties of large serialized trees

→ space trichotomy for path languages:

$O(1)$ $\Theta(\log(n))$ $\Theta(n)$

→ speed-up JSON parsing: RSONPath

Chap 6:
Murlak
Paperman

Incremental complexity: the input changes over time

→ maintain auxiliary data in the RAM model

→ Already known: trichotomy for update time:

$O(1)$ $\Theta(\log(\log(n)))$ $\Theta(\log(n))$

→ Partial extension to tree languages

Chap 7:
Amarilli
Jachiet
Paperman

→ maintain auxiliary data in relational databases

→ Updates with FO formulas

→ Classification of regular languages depending on:

- arity of relations
- complexity of formulas

Chap 7:
Tschirbs
Vortmeier

Conclusion

Regular languages can be optimized in many ways!

Conclusion

Regular languages can be optimized in many ways!

- Circuit complexity: Go deeper in the depth hierarchy (Σ_3, $\mathcal{B}\Sigma_2$, ...)

Conclusion

Regular languages can be optimized in many ways!

- Circuit complexity: Go deeper in the depth hierarchy (Σ_3, $\mathcal{B}\Sigma_2$, ...)
- XML complexity: Extend the trichotomy

Conclusion

Regular languages can be optimized in many ways!

- Circuit complexity: Go deeper in the depth hierarchy (Σ_3, $\mathcal{B}\Sigma_2$, ...)
- XML complexity: Extend the trichotomy
- Incremental complexity: In RAM, finish the trichotomy

Conclusion

Regular languages can be optimized in many ways!

- Circuit complexity: Go deeper in the depth hierarchy (Σ_3, $\mathcal{B}\Sigma_2$, ...)
- XML complexity: Extend the trichotomy
- Incremental complexity: In RAM, finish the trichotomy
- Learning complexity: Study the regular languages in transformer classes
  $\rightarrow$ Postdoc with Thomas Zeume in Bochum

Conclusion

Regular languages can be optimized in many ways!

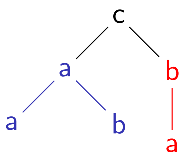
- Circuit complexity: Go deeper in the depth hierarchy (Σ_3, $\mathcal{B}\Sigma_2$, ...)
- XML complexity: Extend the trichotomy
- Incremental complexity: In RAM, finish the trichotomy
- Learning complexity: Study the regular languages in transformer classes
  $\rightarrow$ Postdoc with Thomas Zeume in Bochum

Thanks!

Appendix

XML complexity

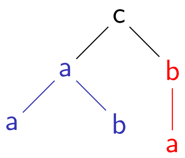
XML encoding of trees:



$\langle c \rangle$
$\quad \langle a \rangle$
$\qquad \langle a \rangle \langle / a \rangle$
$\qquad \langle b \rangle \langle / b \rangle$
$\quad \langle / a \rangle$
$\quad \langle b \rangle$
$\qquad \langle a \rangle \langle / a \rangle$
$\quad \langle / b \rangle$
$\langle / c \rangle$

XML complexity

XML encoding of trees:

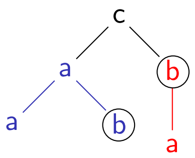


$\langle c\rangle$
$\quad\langle a\rangle$
$\quad\quad\langle a\rangle\langle /a\rangle$
$\quad\quad\langle b\rangle\langle /b\rangle$
$\quad\langle /a\rangle$
$\quad\langle b\rangle$
$\quad\quad\langle a\rangle\langle /a\rangle$
$\quad\langle /b\rangle$
$\langle /c\rangle$

RPQs: the path from the root belongs to a given regular language.

XML complexity

XML encoding of trees:

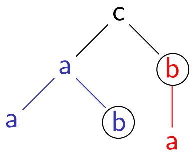


$\langle c\rangle$
$\langle a\rangle$
$\langle a\rangle\langle/a\rangle$
$\langle b\rangle\langle/b\rangle$
$\langle/a\rangle$
$\langle b\rangle$
$\langle a\rangle\langle/a\rangle$
$\langle/b\rangle$
$\langle/c\rangle$

RPQs: the path from the root belongs to a given regular language.
For instance, the RPQ associated to ca^*b.

XML complexity

XML encoding of trees:



$\langle c \rangle$
$\langle a \rangle$
$\langle a \rangle \langle / a \rangle$
$\langle b \rangle \langle / b \rangle$
$\langle / a \rangle$
$\langle b \rangle$
$\langle a \rangle \langle / a \rangle$
$\langle / b \rangle$
$\langle / c \rangle$

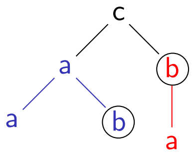
RPQs: the path from the root belongs to a given regular language.
For instance, the RPQ associated to ca^*b.

Thm (B., Murlak, Paperman): Let $\mathcal{L}$ be a regular language. Its RPQ can be executed either in

- Constant memory.
- Logarithmic memory, but not less.
- Linear memory, but not less.

XML complexity

XML encoding of trees:



$\langle c \rangle$
$\quad \langle a \rangle$
$\quad\quad \langle a \rangle \langle / a \rangle$
$\quad\quad \langle b \rangle \langle / b \rangle$
$\quad \langle / a \rangle$
$\quad \langle b \rangle$
$\quad\quad \langle a \rangle \langle / a \rangle$
$\quad \langle / b \rangle$
$\langle / c \rangle$

RPQs: the path from the root belongs to a given regular language.
For instance, the RPQ associated to ca^*b.

Thm (B., Murlak, Paperman): Let $\mathcal{L}$ be a regular language. Its RPQ can be executed either in

- Constant memory.
- Logarithmic memory, but not less.
- Linear memory, but not less.

} Algebraic characterisations

Incremental complexity

a	b	b	a	b	a	a	b

$\in (a+b)^*aa(a+b)^*$

Incremental complexity

a	b	b	a	b	a	b	b

$\notin (a+b)^* aa (a+b)^*$

Incremental complexity

a	b	b	a	b	a	b	a

$\notin (a+b)^*aa(a+b)^*$

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^* aa (a+b)^*$

Incremental complexity

a	b	a	a	b	a	b	a	$\in (a+b)^* aa (a+b)^*$

$\rightarrow$ use auxiliary data structures

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^*aa(a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model

FO formulas

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^* aa (a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)	FO formulas

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^*aa(a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)

Thm (Amarilli, Jachiet, Paperman): A regular language can be maintained in RAM in time either

$O(1)$ or $\Theta(\log(\log(n)))$ or $\Theta(\log(n))$

FO formulas

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^* aa (a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)

Thm (Amarilli, Jachiet, Paperman): A regular language can be maintained in RAM in time either

$O(1)$ or $\Theta(\log(\log(n)))$ or $\Theta(\log(n))$

NEW: characterization of this class for regular languages of trees

FO formulas

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^* aa (a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)

Thm (Amarilli, Jachiet, Paperman): A regular language can be maintained in RAM in time either

$O(1)$ or $\Theta(\log(\log(n)))$ or $\Theta(\log(n))$

NEW: characterization of this class for regular languages of trees

FO formulas

Use relational databases.

Incremental complexity

| a | b | a | a | b | a | b | a | $\in (a+b)^* aa (a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)

Thm (Amarilli, Jachiet, Paperman): A regular language can be maintained in RAM in time either

$O(1)$ or $\Theta(\log(\log(n)))$ or $\Theta(\log(n))$

NEW: characterization of this class for regular languages of trees

FO formulas

Use relational databases.

Binary	Unary

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^* aa (a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)

Thm (Amarilli, Jachiet, Paperman): A regular language can be maintained in RAM in time either

$O(1)$ or $\Theta(\log(\log(n)))$ or $\Theta(\log(n))$

NEW: characterization of this class for regular languages of trees

FO formulas

Use relational databases.

Binary	Unary
Prop $\leftrightarrow$ Reg	

Incremental complexity

$$\boxed{a \mid b \mid a \mid a \mid b \mid a \mid b \mid a} \in (a+b)^* aa (a+b)^*$$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)

Thm (Amarilli, Jachiet, Paperman): A regular language can be maintained in RAM in time either

$O(1)$ or $\Theta(\log(\log(n)))$ or $\Theta(\log(n))$

NEW: characterization of this class for regular languages of trees

FO formulas

Use relational databases.

Binary	Unary
Prop $\leftrightarrow$ Reg	Prop $\leftrightarrow$ Groups

Incremental complexity

a	b	a	a	b	a	b	a

$\in (a+b)^* aa (a+b)^*$

$\rightarrow$ use auxiliary data structures

RAM model (Linear preprocessing)

Thm (Amarilli, Jachiet, Paperman): A regular language can be maintained in RAM in time either

$O(1)$ or $\Theta(\log(\log(n)))$ or $\Theta(\log(n))$

NEW: characterization of this class for regular languages of trees

FO formulas

Use relational databases.

Binary	Unary
Prop $\leftrightarrow$ Reg	Prop $\leftrightarrow$ Groups
	Σ_2 $\leftrightarrow$ Reg
	FO^2 $\leftrightarrow$ Reg