

# Streaming and circuit complexity

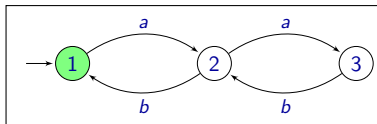
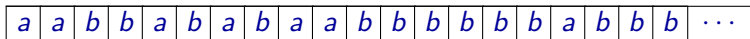
Filip Murlak, [Charles Paperman](#) and Michał Pilipczuk

University of Warsaw, [University of Tübingen](#)

Paris, 2017

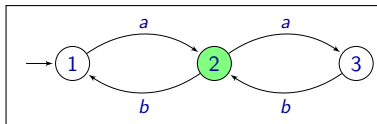
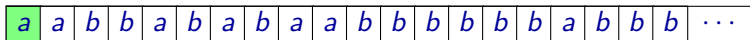
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



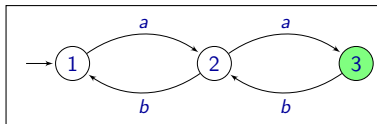
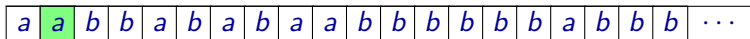
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



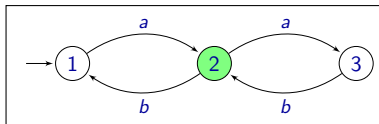
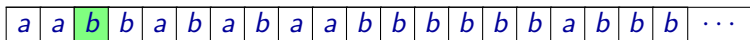
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



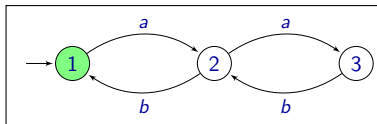
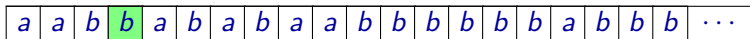
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



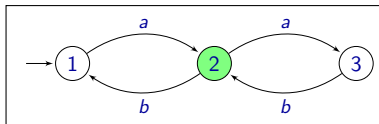
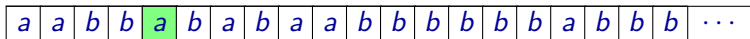
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



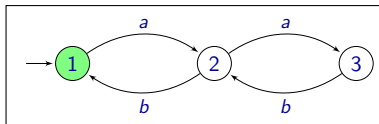
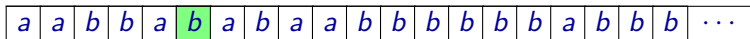
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



A running example:  $(a(ab)^*b)^*$ .

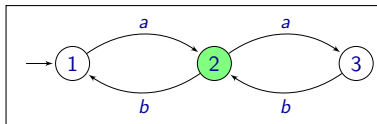
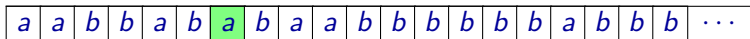
Automata: standard model for streaming with constant memory





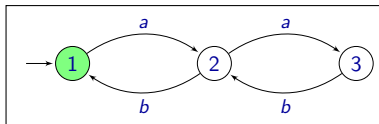
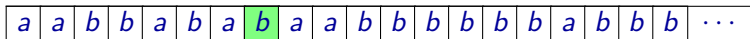
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



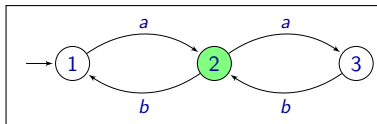
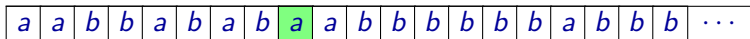
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



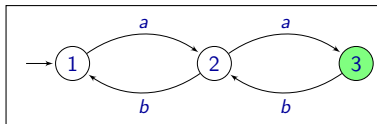
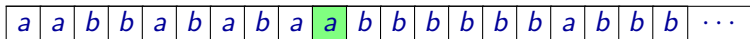
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



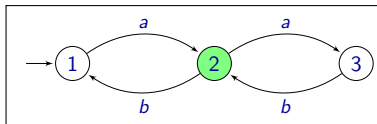
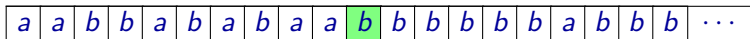
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



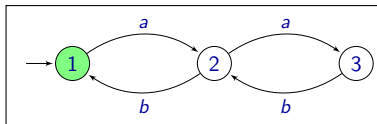
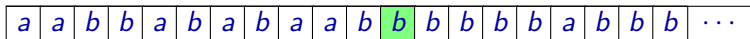
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



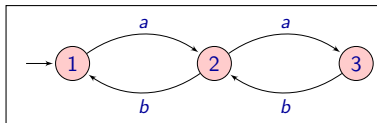
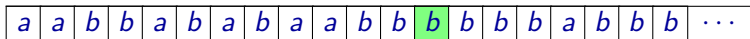
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



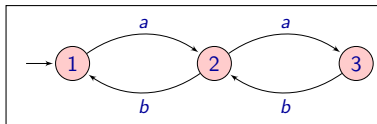
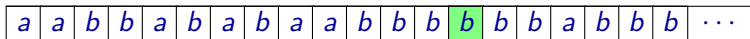
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



A running example:  $(a(ab)^*b)^*$ .

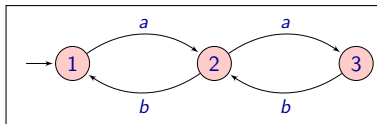
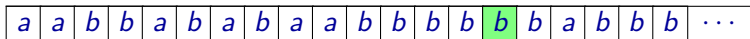
Automata: standard model for streaming with constant memory





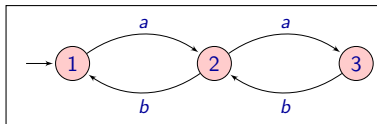
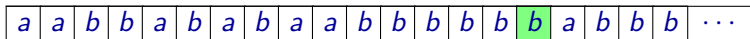
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



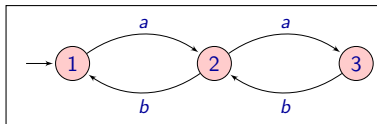
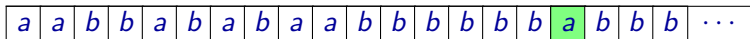
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



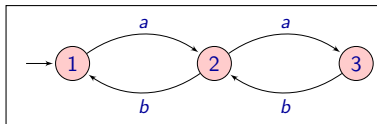
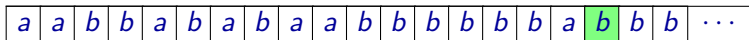
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



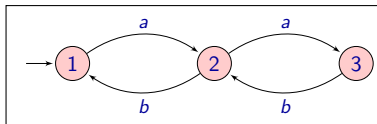
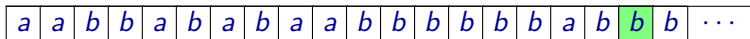
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



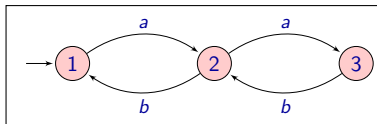
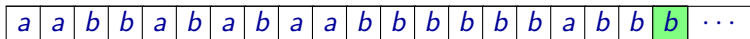
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



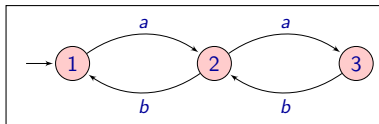
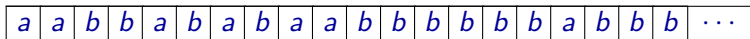
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



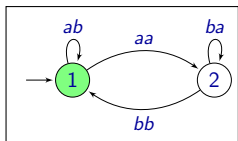
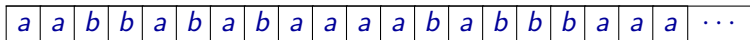
A running example:  $(a(ab)^*b)^*$ .

Automata: standard model for streaming with constant memory



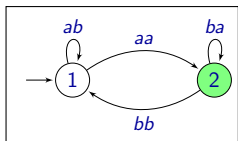
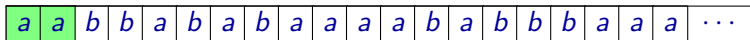
→ However, data is not streamed bit by bit but by **block**.

A running example:  $(a(ab)^*b)^*$ .

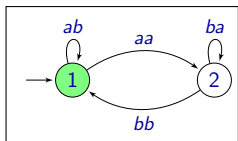
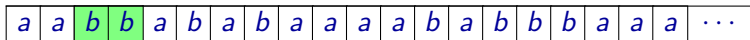




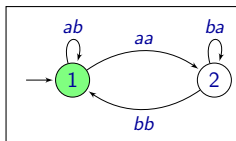
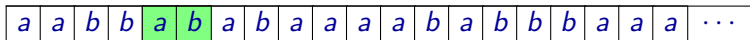
A running example:  $(a(ab)^*b)^*$ .



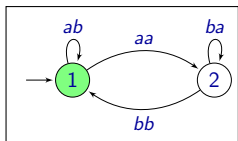
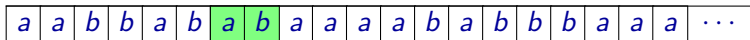
A running example:  $(a(ab)^*b)^*$ .



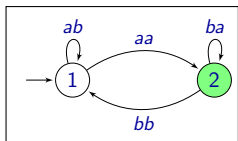
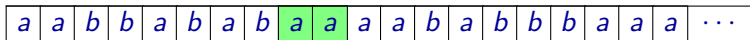
A running example:  $(a(ab)^*b)^*$ .



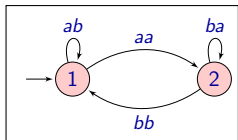
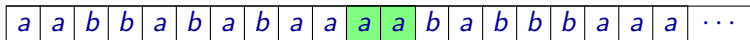
A running example:  $(a(ab)^*b)^*$ .



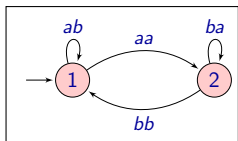
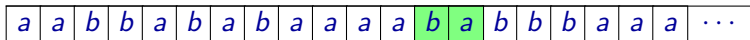
A running example:  $(a(ab)^*b)^*$ .



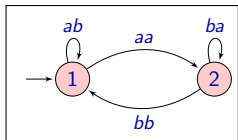
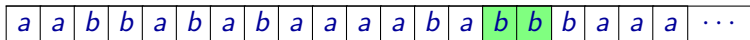
A running example:  $(a(ab)^*b)^*$ .



A running example:  $(a(ab)^*b)^*$ .

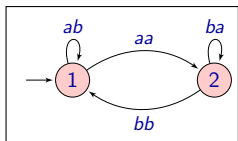
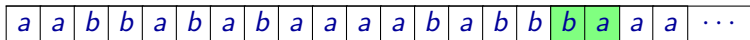


A running example:  $(a(ab)^*b)^*$ .

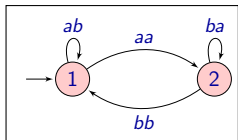
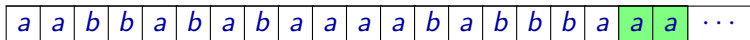




A running example:  $(a(ab)^*b)^*$ .

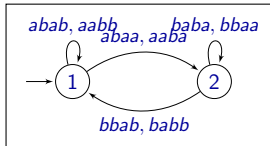


A running example:  $(a(ab)^*b)^*$ .

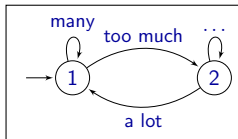
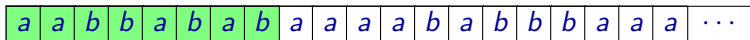


A running example:  $(a(ab)^*b)^*$ .

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



A running example:  $(a(ab)^*b)^*$ .



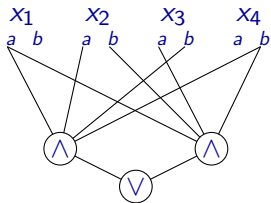
→ Number of transitions grows **exponentially** with the block size.

How to describe automata over block of letters ?

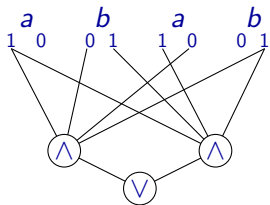
## How to describe automata over block of letters ?

1. A new model for local parallelisation: the **streaming circuits**.
2. **Complexity** notions for this model.
3. An application to **XML-schema**.

# Circuits

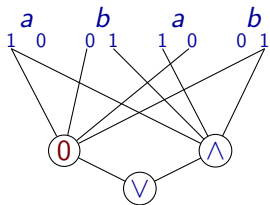


# Circuits

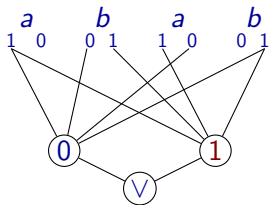




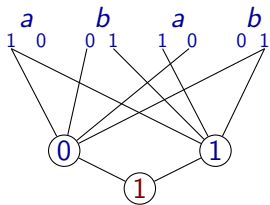
# Circuits



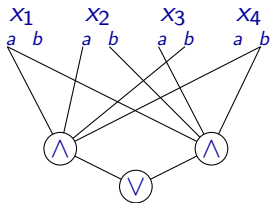
# Circuits



# Circuits

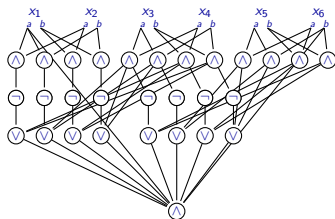


# Circuits



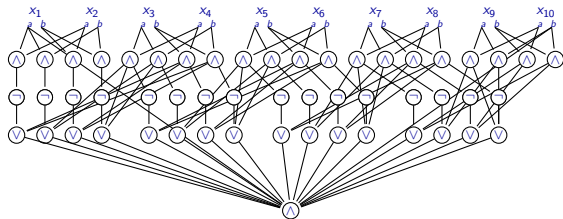
→ Recognizes  $(a(ab)^*b)^* \cap A^4 = \{abab, aabb\}$ .

# Circuits family



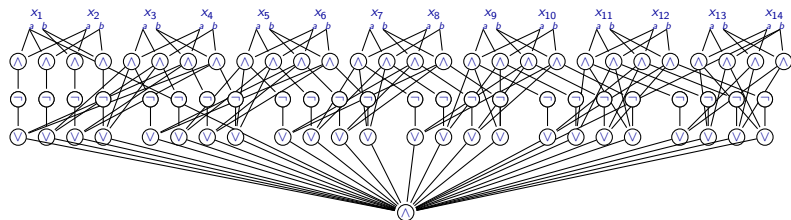
→ Recognizes  $(a(ab)^*b)^* \cap A^6$ .

# Circuits family



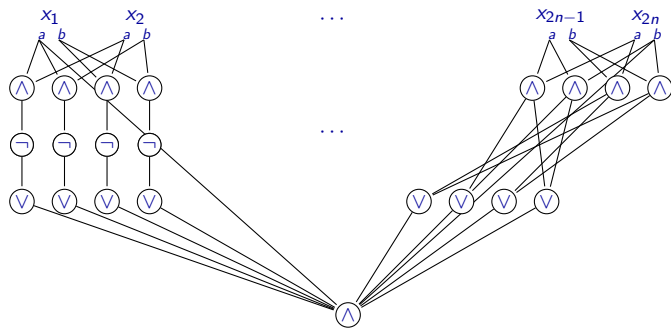
→ Recognizes  $(a(ab)^*b)^* \cap A^{10}$ .

# Circuits family



→ Recognizes  $(a(ab)^*b)^* \cap A^{14}$ .

# Circuits family

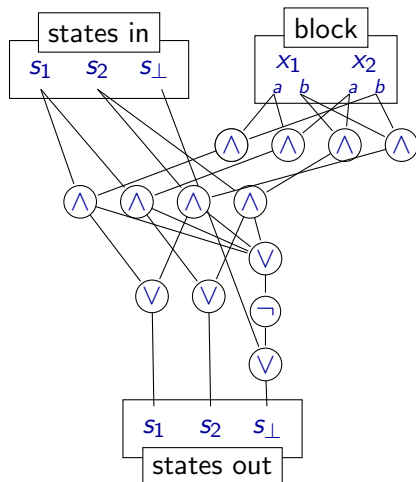
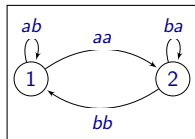


→ Recognizes  $(a(ab)^*b)^* \cap A^{2n}$  with **const.** depth and **linear** size.

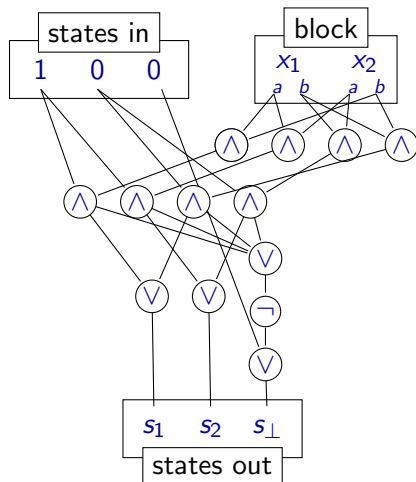
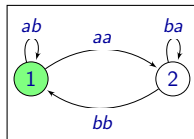
→ **Non uniform** model of computation.



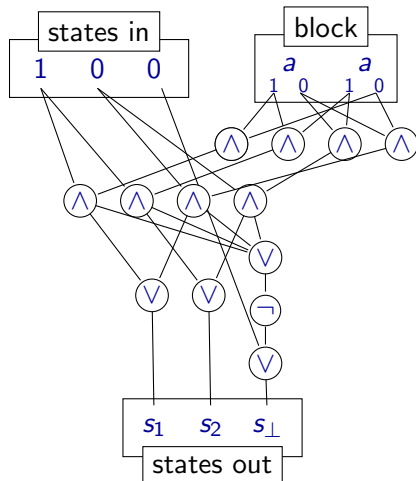
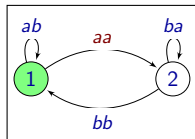
# The streaming circuits



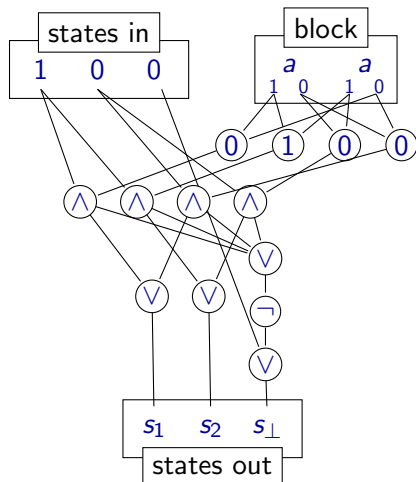
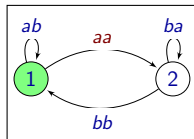
# The streaming circuits



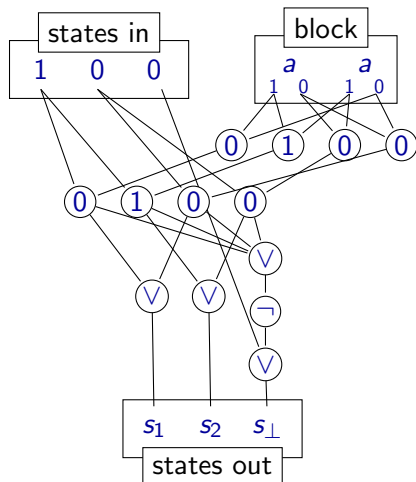
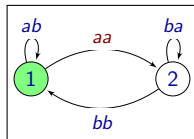
# The streaming circuits



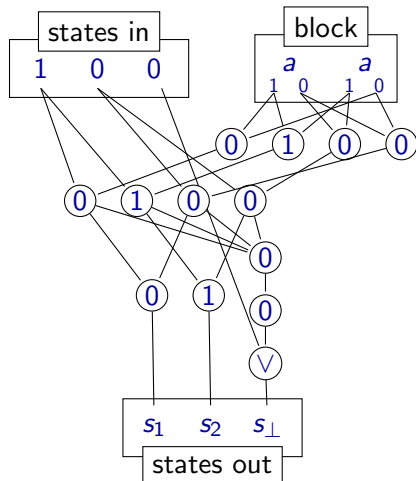
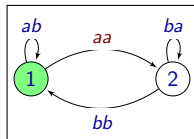
# The streaming circuits



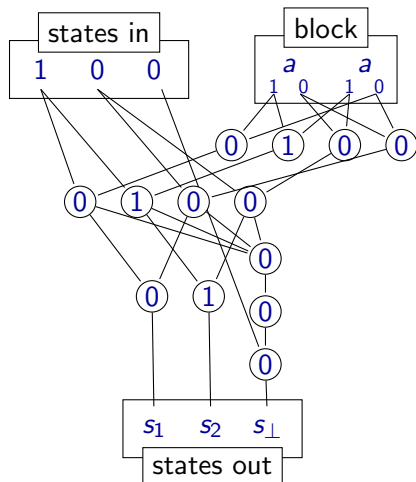
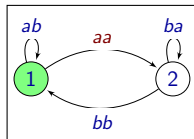
# The streaming circuits



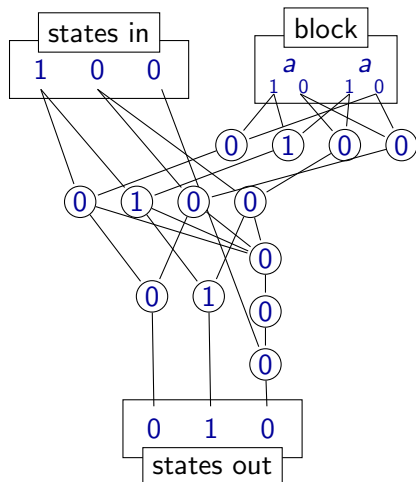
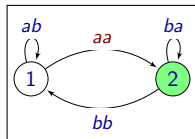
# The streaming circuits



# The streaming circuits

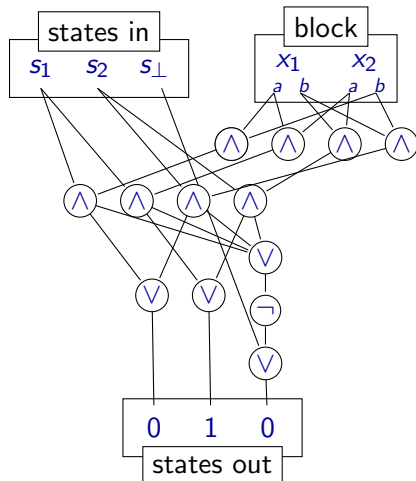
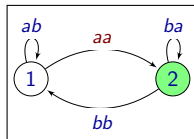


# The streaming circuits

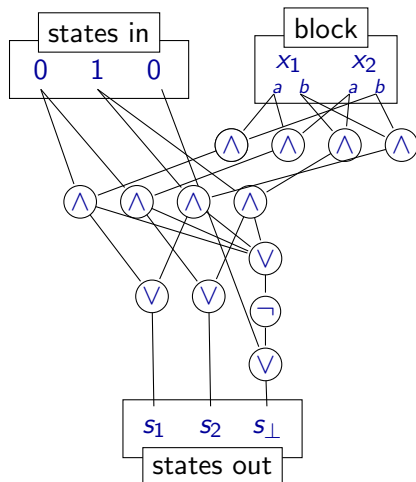
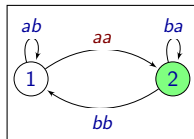




# The streaming circuits



# The streaming circuits

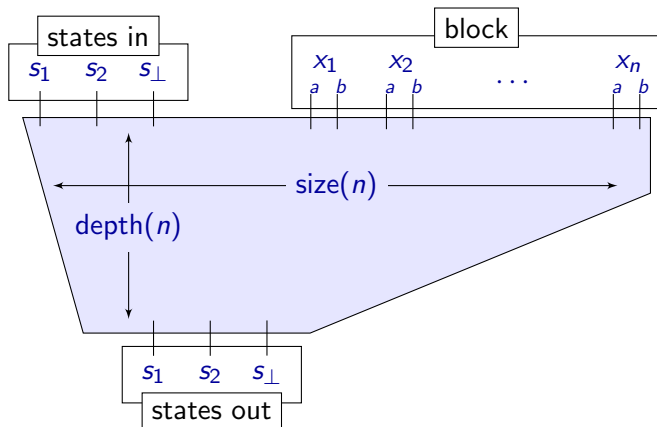


## The block complexity

→ Growth of #gates, of #wires and the depth w.r.t. block length?

# The block complexity

→ Growth of #gates, of #wires and the depth w.r.t. block length?



# Complexity

Theorem (Murlak, P., Pilipczuk, 2016).

For regular languages, the **block** complexity is equivalent to the **circuit** complexity.

# Complexity

Theorem (Murlak, P., Pilipczuk, 2016).

For regular languages, the **block** complexity is equivalent to the **circuit** complexity.

→ We can now use **30** years of research on the circuit complexity of regular languages.

# Goals

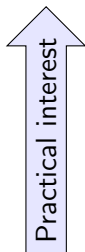
- (1) Describe languages with **good** complexity.
- (2) Decide **effectively** membership to these classes.
- (3) Prove **optimality** of these classes.

# Goals

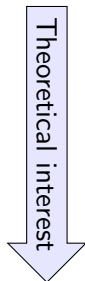
- (1) Descriptive complexity of classes of circuits.
- (2) Logical definability of regular languages.
- (3) Circuit complexity lower bounds.



# Goals



- (1) **Descriptive complexity** of classes of circuits.
- (2) **Logical definability** of regular languages.
- (3) Circuit complexity **lower bounds**.



# NC<sup>1</sup>-complexity

Proposition (Folklore).

A regular language computable by an automaton with  $q$  states has a block complexity  $\log n$  depth and size  $\mathcal{O}(q^3 n)$ .

- A simple divide and conquer.
- Provides already nice hardware synthesis.

# NC<sup>1</sup>-complexity

Proposition (Folklore).

A regular language computable by an automaton with  $q$  states has a block complexity  $\log n$  depth and size  $\mathcal{O}(q^3 n)$ .

- A simple divide and conquer.
- Provides already nice hardware synthesis.

Can we do better ?

# NC<sup>1</sup>-complexity

Proposition (Folklore).

A regular language computable by an automaton with  $q$  states has a block complexity  $\log n$  depth and size  $\mathcal{O}(q^3 n)$ .

- A simple divide and conquer.
- Provides already nice hardware synthesis.

Can we do better ?

Theorem (Barrington, 1989).

Regular languages are complete for **NC<sup>1</sup>**.

# NC<sup>1</sup>-complexity

Proposition (Folklore).

A regular language computable by an automaton with  $q$  states has a block complexity  $\log n$  depth and size  $\mathcal{O}(q^3 n)$ .

- A simple divide and conquer.
- Provides already nice hardware synthesis.

Can we do better ?

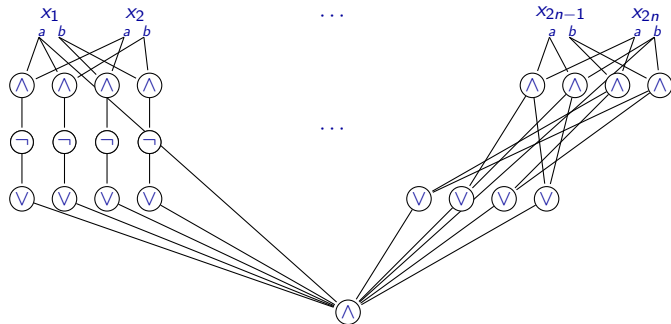
No ! :(

Theorem (Barrington, 1989).

Regular languages are complete for NC<sup>1</sup>.

Back to  $(a(ab)^*b)^*$

→ It has a **constant** depth and **linear** size.



When can we improve the **NC<sup>1</sup>** construction ?

# $AC^0$ -complexity

Theorem (Barrington et al., 1992).

Regular languages computable with **const** depth and **polysize** are exactly those definable in **first order** logic with **regular predicates**.

# $AC^0$ -complexity

Theorem (Barrington et al., 1992).

Regular languages computable with **const** depth and **polysize** are exactly those definable in **first order** logic with **regular predicates**. Moreover this class has a **decidable** membership.



# $AC^0$ -complexity

Theorem (Barrington et al., 1992).

Regular languages computable with **const** depth and **polysize** are exactly those definable in **first order** logic with **regular predicates**. Moreover this class has a **decidable** membership.

→ it is a consequence of two important results:

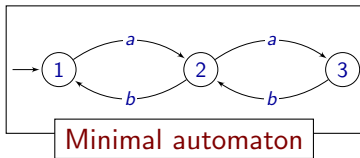
1. **Parity** is not in  $AC^0$ .

Furst, Saxe and Sipser (1984)

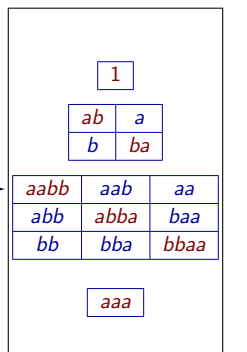
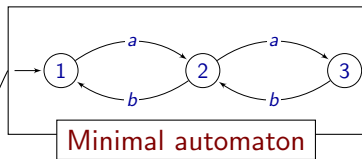
2. For regular languages, **first-order** definability is equivalent to **aperiodicity**.

Schützenberger (1965), McNaughton and Papert (1971).

Back to  $(a(ab)^*b)^*$

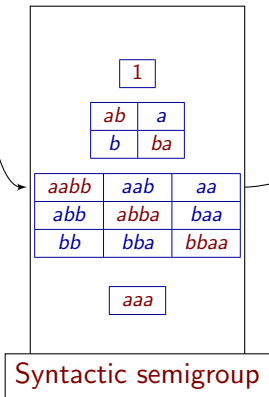
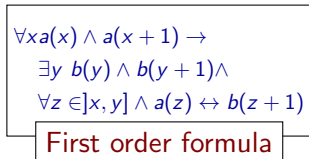
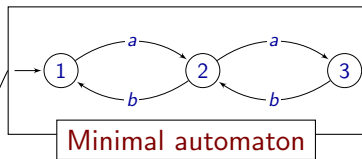


Back to  $(a(ab)^*b)^*$

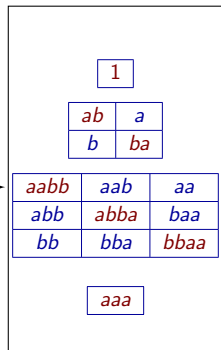
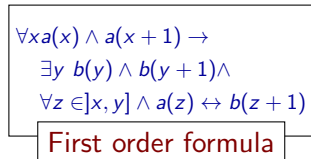
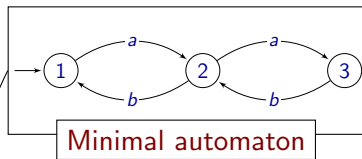


<i>aabb</i>	<i>aab</i>	<i>aa</i>
<i>abb</i>	<i>abba</i>	<i>baa</i>
<i>bb</i>	<i>bba</i>	<i>bbaa</i>

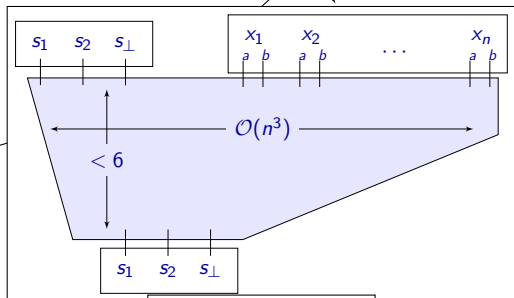
# Back to $(a(ab)^*b)^*$



# Back to $(a(ab)^*b)^*$

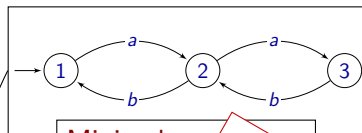


Syntactic semigroup



Streaming circuit

# Back to $(a(ab)^*b)^*$



Minimal automaton

$\forall x a(x) \wedge a(x+1) \rightarrow$   
 $\exists y b(y) \wedge b(y+1) \wedge$   
 $\forall z \in [x, y] \wedge a(z) \leftrightarrow b(z+1)$

First order formula

Not the best !!

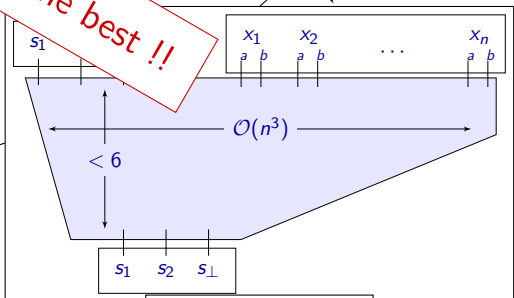
1

ab	a
b	ba

aabb	aab	aa
abb	abba	baa
bb	bba	bbaa

aaa

Syntactic semigroup



Streaming circuit

# Size and depth tradeoff

## The CFL procedure

For regular languages, it is possible to decrease the size by doubling the depth.

→ All regular languages of  $\mathbf{AC}^0$  are computable by circuits with wire-size in  $\mathcal{O}(n \log^*(n))$ .

# Size and depth tradeoff

## The CFL procedure

For regular languages, it is possible to decrease the size by doubling the depth.

→ All regular languages of  $AC^0$  are computable by circuits with wire-size in  $\mathcal{O}(n \log^*(n))$ .

Theorem (Barrington et al. 1992).

Regular languages computable with **const** depth and **polynomial** size are exactly those definable in **first order** logic with **regular predicates**. Moreover this class has a **decidable** membership.



# Size and depth tradeoff

## The CFL procedure

For regular languages, it is possible to decrease the size by doubling the depth.

→ All regular languages of  $AC^0$  are computable by circuits with wire-size in  $\mathcal{O}(n \log^*(n))$ .

## Theorem

Regular languages computable with **const** depth and ~~polynomial~~ **wire linear** size are exactly those definable in **two variable first order** logic with **regular predicates**. Moreover this class has a **decidable** membership.

# Size and depth tradeoff

## The CFL procedure

For regular languages, it is possible to decrease the size by doubling the depth.

→ All regular languages of  $AC^0$  are computable by circuits with wire-size in  $\mathcal{O}(n \log^*(n))$ .

## Theorem

Regular languages computable with **const** depth and ~~polynomial~~ **wire linear** size are exactly those definable in **two variable first order** logic with **regular predicates**. Moreover this class has a **decidable** membership.

# Size and depth tradeoff

## The CFL procedure

For regular languages, it is possible to decrease the size by doubling the depth.

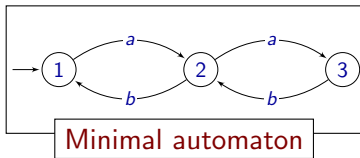
→ All regular languages of  $AC^0$  are computable by circuits with wire-size in  $\mathcal{O}(n \log^*(n))$ .

## Theorem

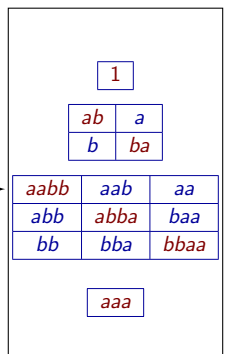
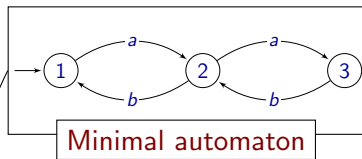
Regular languages computable with **const** depth and ~~polynomial~~ **wire linear** size are exactly those definable in **two variable first order** logic with **regular predicates**. Moreover this class has a **decidable** membership.

→ Still open for **gate linear** complexity, but probably very difficult.

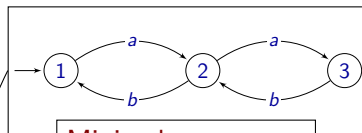
Back to  $(a(ab)^*b)^*$



Back to  $(a(ab)^*b)^*$



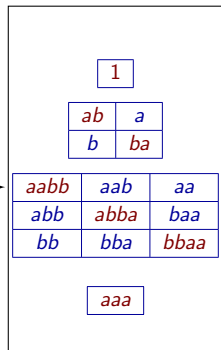
Back to  $(a(ab)^*b)^*$



Minimal automaton

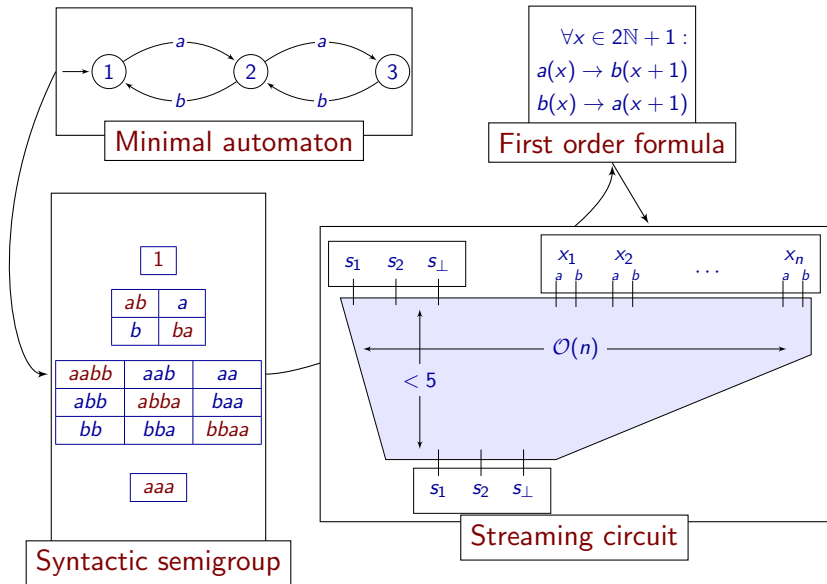
$\forall x \in 2\mathbb{N} + 1 :$   
 $a(x) \rightarrow b(x + 1)$   
 $b(x) \rightarrow a(x + 1)$

First order formula

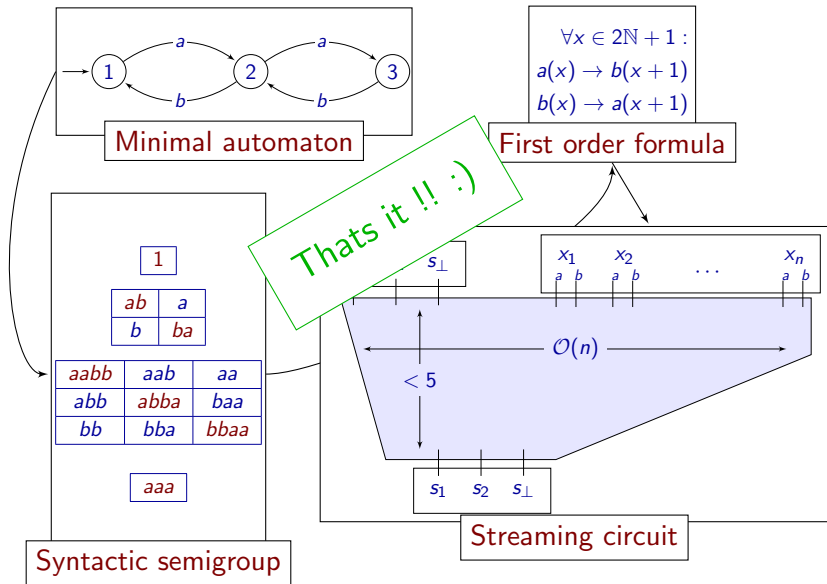


Syntactic semigroup

# Back to $(a(ab)^*b)^*$



# Back to $(a(ab)^*b)^*$





# Depth and size tradeoff

## Conjecture (Straubing, 1994)

The regular languages computable by streaming circuits of depth  $d$  and **polysize** are those definable in the  $d^{\text{th}}$  **alternation hierarchy** of first order logic with regular predicates.

# Depth and size tradeoff

## Conjecture (Straubing, 1994)

The regular languages computable by streaming circuits of depth  $d$  and **polysize** are those definable in the  $d^{\text{th}}$  **alternation hierarchy** of first order logic with regular predicates.

→ Decidability **membership** in these classes is one of the most important questions of automata theory.

What is the block complexity of checking that an XML document satisfies a DTD ?

What is the block complexity of checking that an XML document satisfies a non-recursive DTD ?

→ Circuit complexity of unbounded regular tree languages is widely unknown.

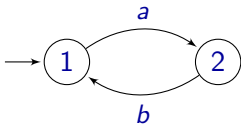
What is the block complexity of checking that an XML document satisfies a non-recursive DTD ?

→ Circuit complexity of unbounded regular tree languages is widely unknown.

Theorem (Murlak, P., Pilipczuk, 2016)

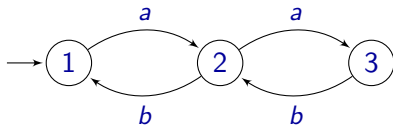
A first order definable non recursive DTD is computable by polysize and constant depth circuits.

# The Dyck languages of constant depth



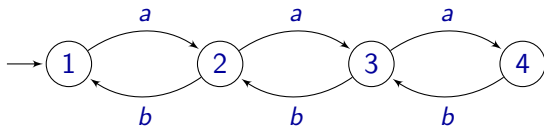
$(ab)^*$

# The Dyck languages of constant depth



$$(a(ab)^*b)^*$$

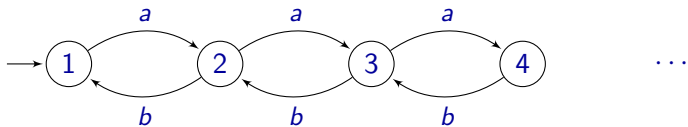
# The Dyck languages of constant depth



$$(a(a(ab)^*b)^*b)^*$$

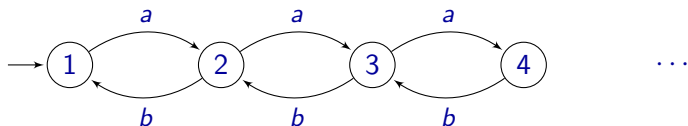


# The Dyck languages of constant depth



$$\begin{cases} L_0 = \epsilon \\ L_{i+1} = (aL_i b)^* \end{cases}$$

# The Dyck languages of constant depth

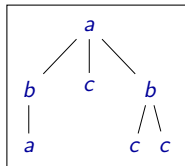


$$\begin{cases} L_0 = \epsilon \\ L_{i+1} = (aL_i b)^* \end{cases}$$

- Each  $L_i$  is the trivial “DTD” of depth  $i$ .
- Separate the alternation hierarchy of first order logic.
- Probably separate the depth hierarchy as well.

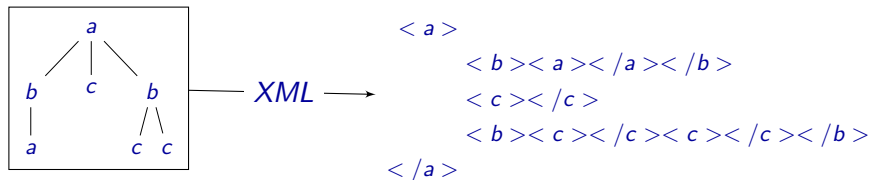
## Encoding of trees

→ XML documents are **standard** encoding of trees.



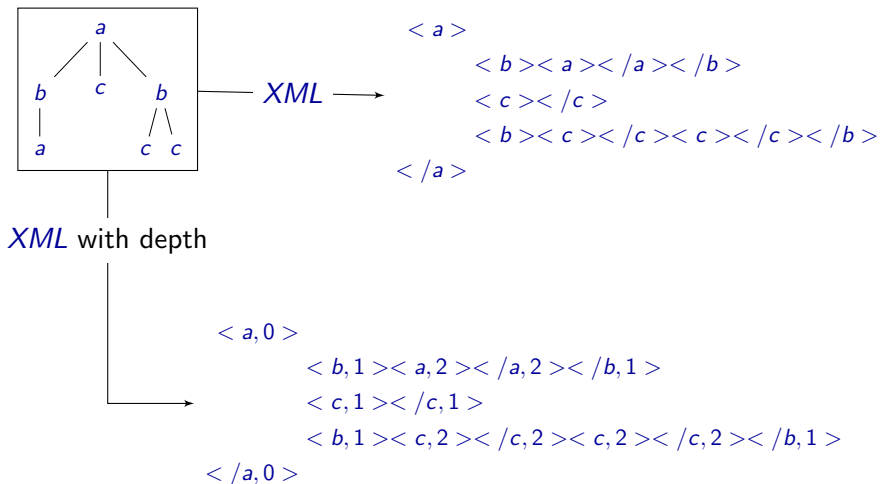
## Encoding of trees

→ XML documents are **standard** encoding of trees.



## Encoding of trees

→ XML documents are **standard** encoding of trees.



# Simple DTD

→ **Simple DTD** form a classical subset of DTD where productions are very **constrained**.

# Simple DTD

→ **Simple DTD** form a classical subset of DTD where productions are very **constrained**.

Theorem (Murlak, P., Pilipczuk, 2016).

**Non recursive simple DTD** over XML with depth are computable by a streaming circuit of depth **3** and with **linearly many wire**.

# Simple DTD

→ **Simple DTD** form a classical subset of DTD where productions are very **constrained**.

Theorem (Murlak, P., Pilipczuk, 2016).

**Non recursive simple DTD** over XML with depth are computable by a streaming circuit of depth **3** and with **linearly many wire**.

→ Constants are **small enough** to consider implementation.



## And now...

→ Some difficult open questions:

- ▶ Is gate linear the same as wire linear for regular languages ?  
(P., 2015)
- ▶ How to prove the Straubing conjecture(s) ?  
(No clue)

→ Some modelisation questions:

- ▶ How to extend to **transducers** ?  
(Cadilhac, Krebs, Ludwig and P., 2015)
- ▶ How to extend to **trees** ?  
(Murlak, P., Pilipczuk, 2016)
- ▶ How to throw in some extra **memory** ?  
(On going investigation with Cadilhac)

# The automata compilation roadmap

**Input:** An automaton.

**Output:** A ready to be implemented **optimal** streaming circuit.

# The automata compilation roadmap

**Input:** An automaton.

**Output:** A ready to be implemented **optimal** streaming circuit.

- ▶ Can you describe this automaton in first order logic ?
- **YES.** Are the numbers of alternation and variables **small** ?
  - **YES.** Then enjoy your small **streaming circuit**.
  - **NO.** You can still play with the **size-depth** tradeoff algorithms to produce a better design in constant depth.
- **IDK** You can run **first order** definability algorithm to check it.
- **NO** Use the **divide and conquer** construction. There is no constant depth circuit in this case.