Streaming and circuit complexity

Filip Murlak, Charles Paperman and Michał Pilipczuk

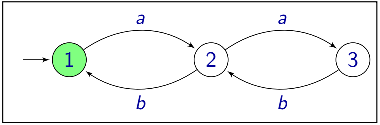
University of Warsaw, University of Tübingen

Paris, 2017

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

Automata: standard model for streaming with constant memory

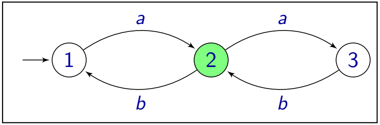
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

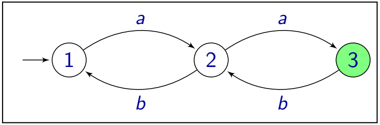
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

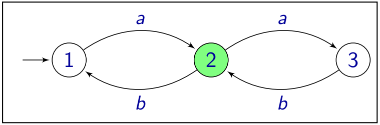
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

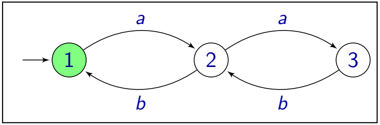
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

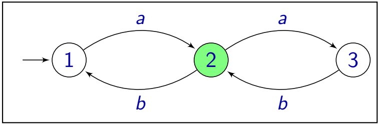
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

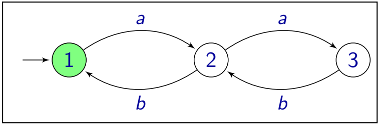
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

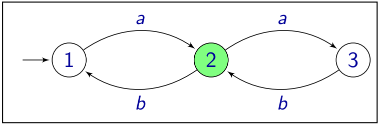
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

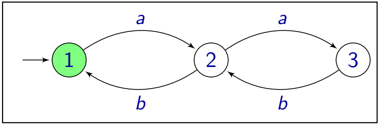
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

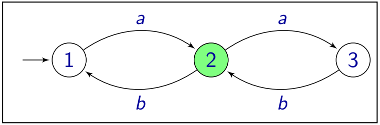
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

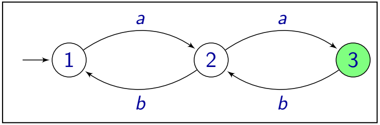
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

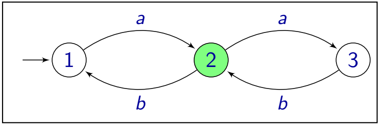
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

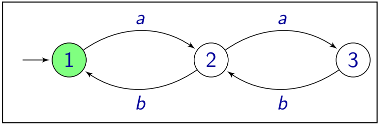
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

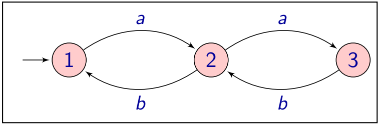
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

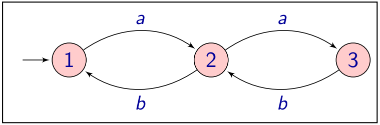
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

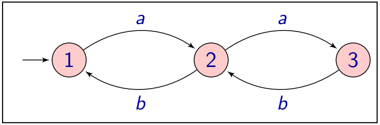
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

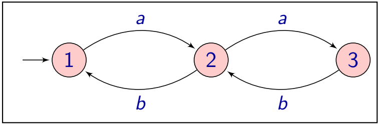
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

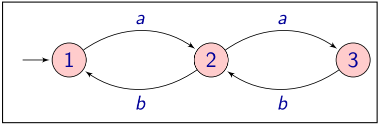
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

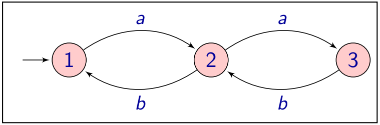
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

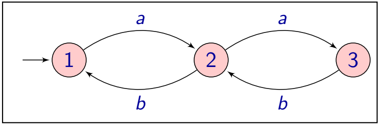
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

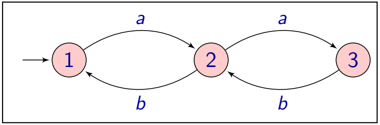
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

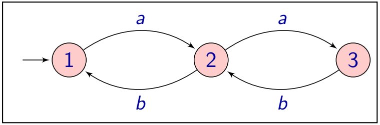
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



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

Automata: standard model for streaming with constant memory

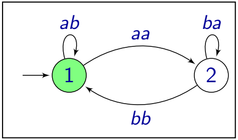
a	a	b	b	a	b	a	b	a	a	b	b	b	b	b	b	a	b	b	b	$\cdots$



$\rightarrow$ However, data is not streamed bit by bit but by block.

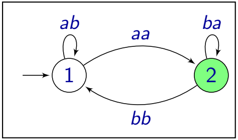
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



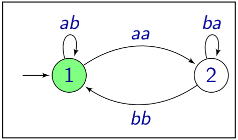
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



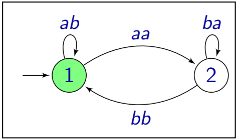
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



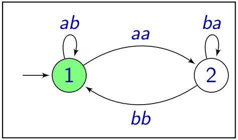
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



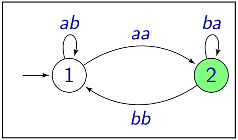
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



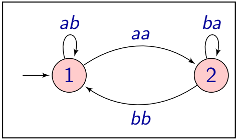
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



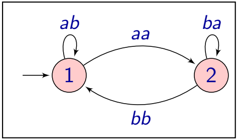
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



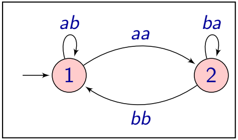
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



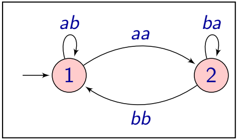
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



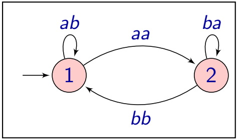
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



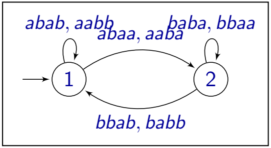
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



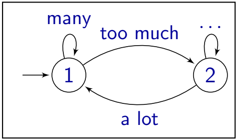
A running example: $(a(ab)^*b)^*$.

a	a	b	b	a	b	a	b	a	a	a	a	b	a	b	b	b	a	a	a	$\cdots$



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	$\cdots$



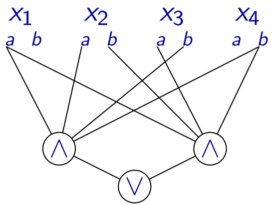
$\rightarrow$ Number of transitions grows exponentially with the block size.

How to describe automata over block of letters ?

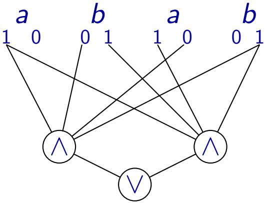
How to describe automata over block of letters ?

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

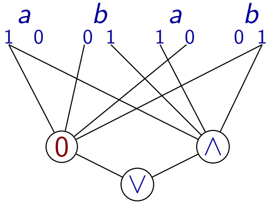
Circuits



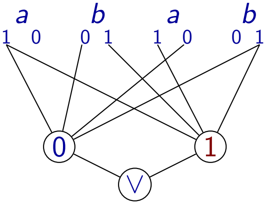
Circuits



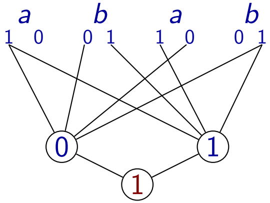
Circuits



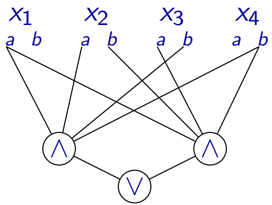
Circuits



Circuits

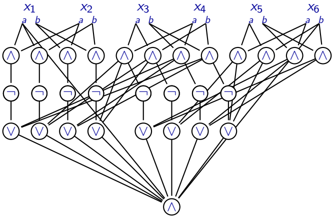


Circuits



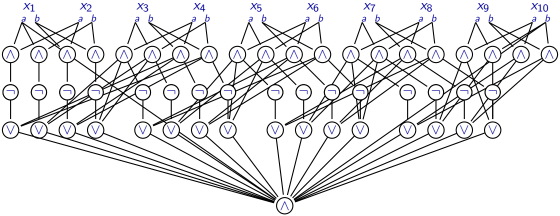
$\rightarrow$ Recognizes $(a(ab)^*b)^* \cap A^4 = \{abab, aabb\}$.

Circuits familly



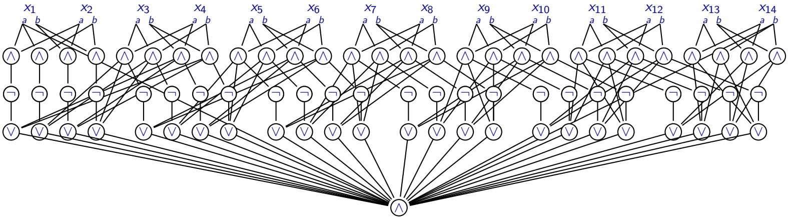
$\rightarrow$ Recognizes $(a(ab)^*b)^* \cap A^6$.

Circuits familly



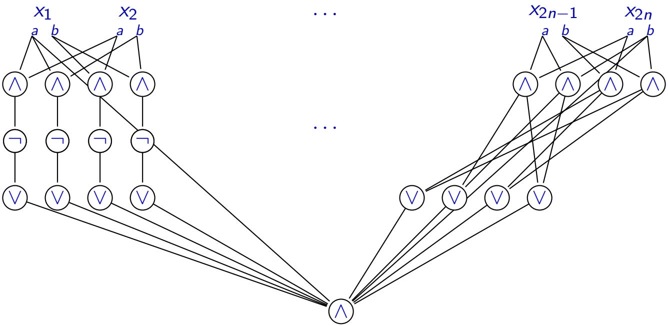
$\rightarrow$ Recognizes $(a(ab)^*b)^* \cap A^{10}$.

Circuits familly



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

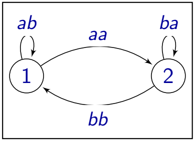
Circuits familly

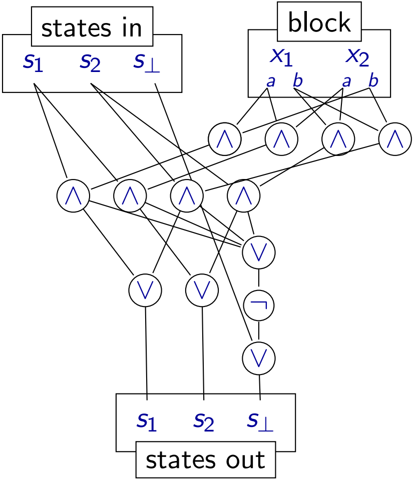


$\rightarrow$ Recognizes $(a(ab)^*b)^* \cap A^{2n}$ with const. depth and linear size.

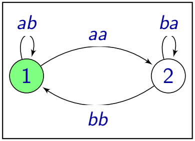
$\rightarrow$ Non uniform model of computation.

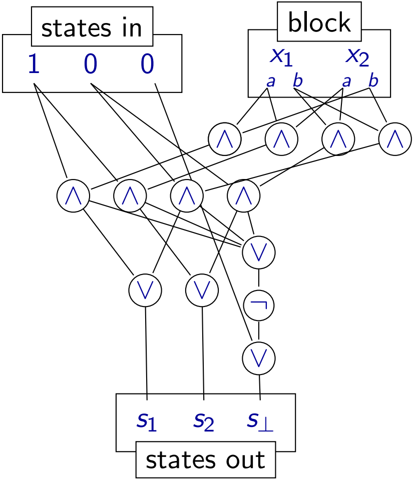
The streaming circuits



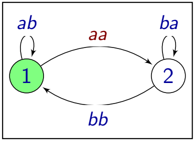


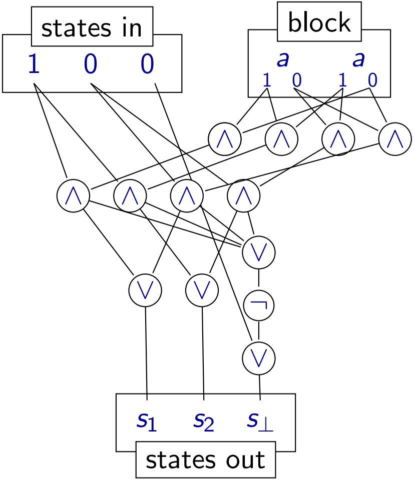
The streaming circuits



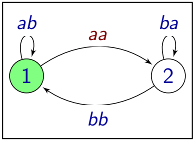


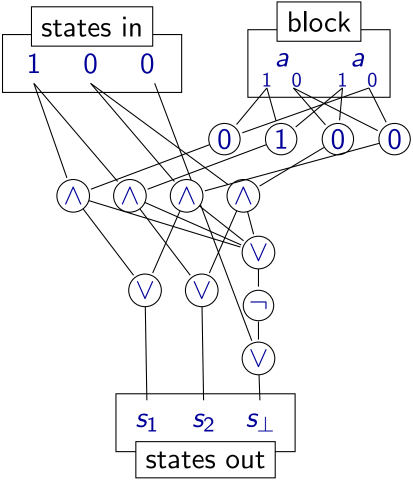
The streaming circuits



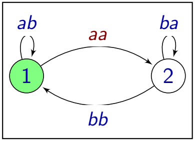


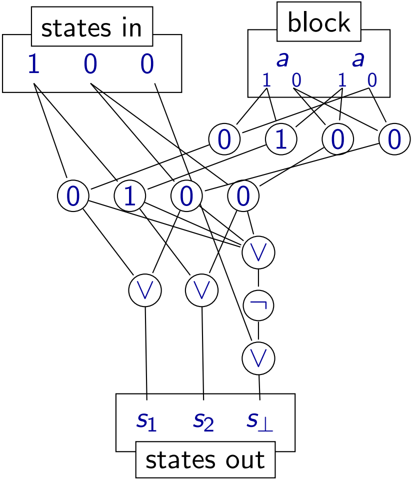
The streaming circuits



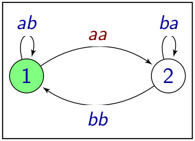


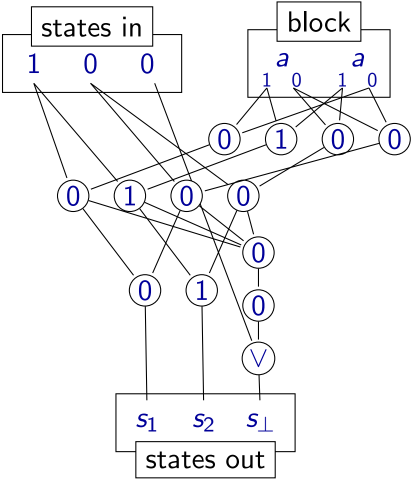
The streaming circuits



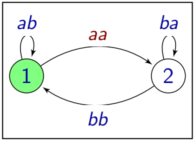


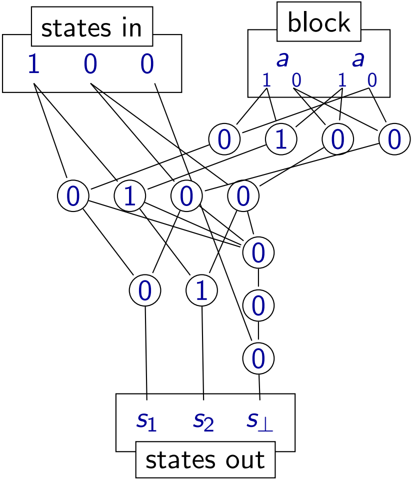
The streaming circuits



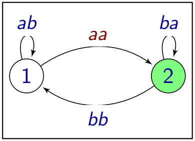


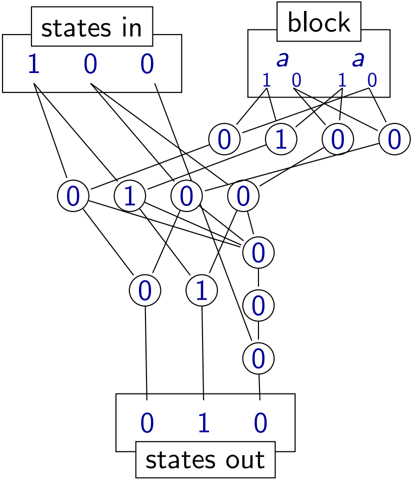
The streaming circuits



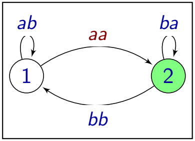


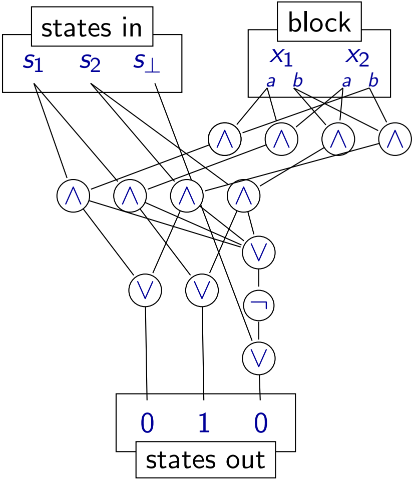
The streaming circuits



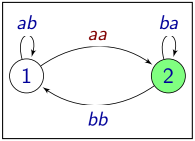


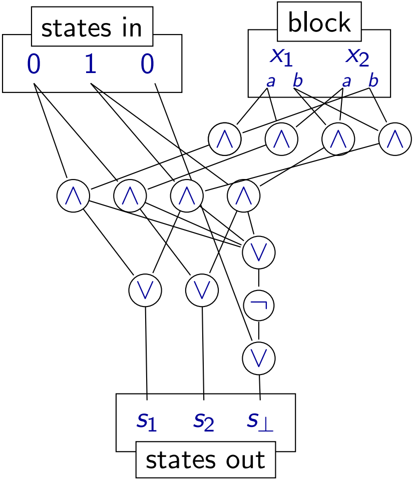
The streaming circuits





The streaming circuits



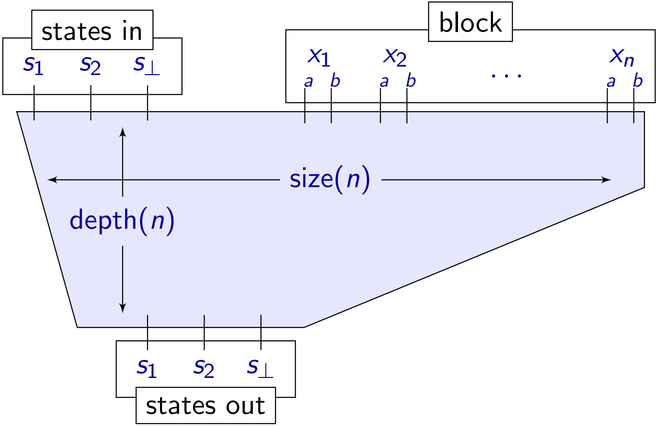


The block complexity

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

The block complexity

$\rightarrow$ 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.

$\rightarrow$ 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

Practical interest

(1) Descriptive complexity of classes of circuits.

(2) Logical definability of regular languages.

(3) Circuit complexity lower bounds.

Theoretical interest

$\mathbf{NC^1}$-complexity

Proposition (Folklore).

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

$\rightarrow$ A simple divide and conquer.
$\rightarrow$ Provides already nice hardware synthesis.

$\mathbf{NC^1}$-complexity

Proposition (Folklore).

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

$\rightarrow$ A simple divide and conquer.
$\rightarrow$ Provides already nice hardware synthesis.

Can we do better ?

$\mathbf{NC^1}$-complexity

Proposition (Folklore).

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

$\rightarrow$ A simple divide and conquer.
$\rightarrow$ Provides already nice hardware synthesis.

Can we do better ?

Theorem (Barrington, 1989).

Regular languages are complete for $\mathbf{NC^1}$.

$\mathbf{NC^1}$-complexity

Proposition (Folklore).

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

$\rightarrow$ A simple divide and conquer.
$\rightarrow$ Provides already nice hardware synthesis.

Can we do better ?

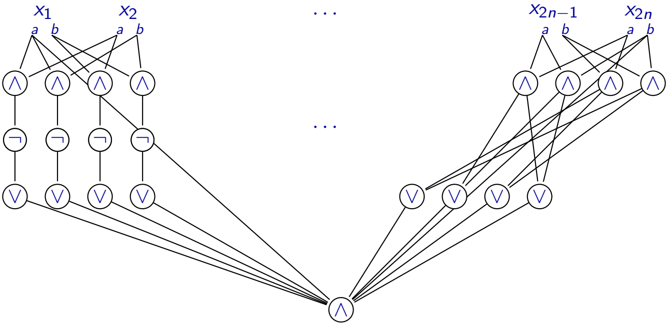
No ! :(

Theorem (Barrington, 1989).

Regular languages are complete for $\mathbf{NC^1}$.

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

$\rightarrow$ It has a constant depth and linear size.



When can we improve the $\mathbf{NC^1}$ construction ?

$\mathbf{AC^0}$-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.

$\mathbf{AC^0}$-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.

$\mathbf{AC^0}$-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.

$\rightarrow$ it is a consequence of two importants results:

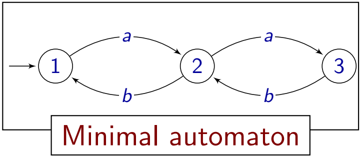
1. Parity is not in $\mathbf{AC^0}$.

   Furst, Saxe and Sipser (1984)

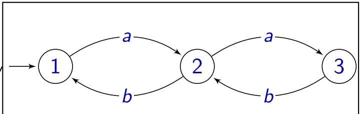
2. For regular languages, first-order definability is equivalent to aperiodicity.

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

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



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



Minimal automaton

1

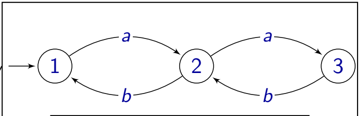
ab	*a*
b	*ba*

aabb	*aab*	*aa*
abb	*abba*	*baa*
bb	*bba*	*bbaa*

aaa

Syntactic semigroup

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



Minimal automaton

$$
\begin{aligned}
&\forall x a(x) \wedge a(x+1) \rightarrow \\
&\quad \exists y \ b(y) \wedge b(y+1) \wedge \\
&\quad \forall z \in]x, y] \wedge a(z) \leftrightarrow b(z+1)
\end{aligned}
$$

First order formula

1

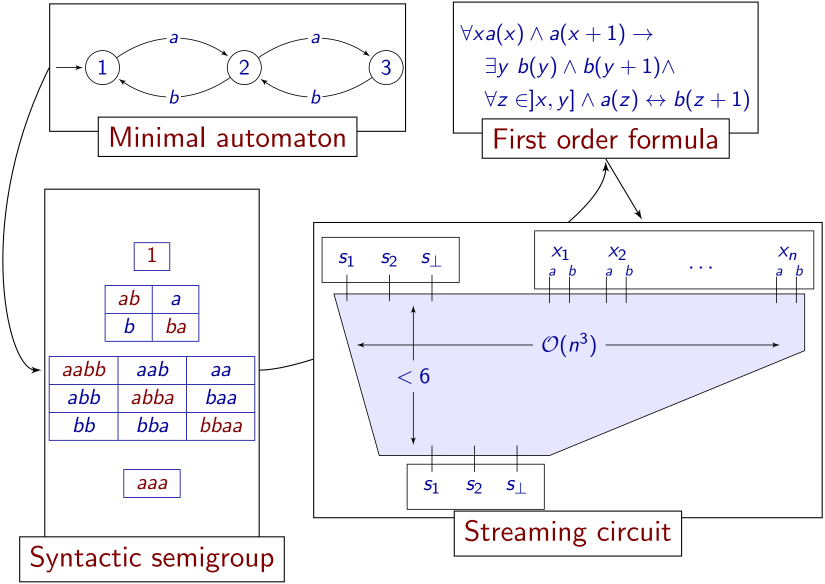
ab	*a*
b	*ba*

aabb	*aab*	*aa*
abb	*abba*	*baa*
bb	*bba*	*bbaa*

aaa

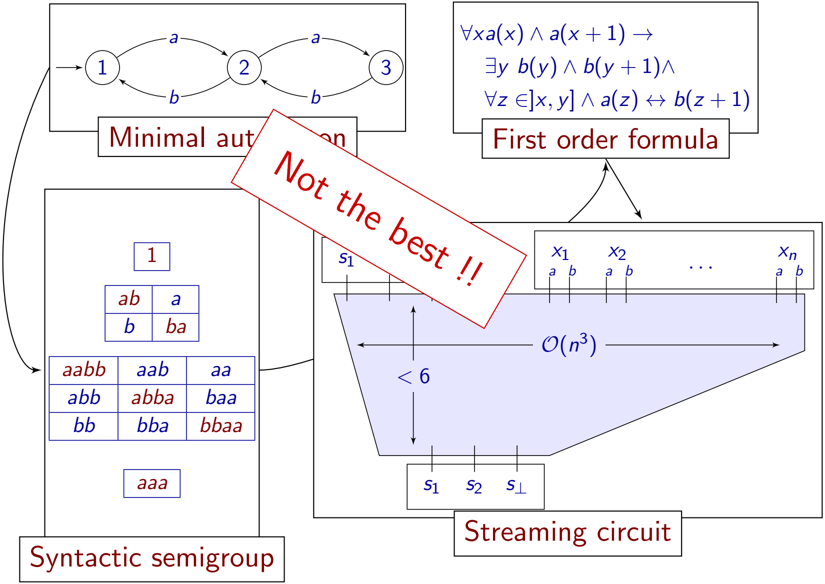
Syntactic semigroup

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



1
2
3
a
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
1
ab
a
b
ba
aabb
aab
aa
abb
abba
baa
bb
bba
bbaa
aaa
Syntactic semigroup
s_1
s_2
$s_\perp$
x_1
x_2
$\dots$
x_n
$\mathcal{O}(n^3)$
< 6
Streaming circuit

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



Minimal automaton
First order formula
$\forall x\, a(x) \wedge a(x+1) \rightarrow$
$\exists y\ b(y) \wedge b(y+1) \wedge$
$\forall z \in]x,y] \wedge a(z) \leftrightarrow b(z+1)$
Syntactic semigroup
Streaming circuit
$\mathcal{O}(n^3)$
< 6
Not the best !!

Size and depth tradeoff

The CFL procedure

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

$\rightarrow$ 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.

$\rightarrow$ All regular languages of $\mathbf{AC^0}$ 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 polynormial 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.

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

Theorem

Regular languages computable with const depth and ~~polynormial~~ 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.

$\rightarrow$ All regular languages of $\mathbf{AC^0}$ 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.

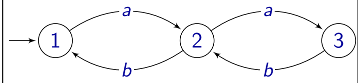
$\rightarrow$ All regular languages of $\mathbf{AC^0}$ are computable by circuits with wire-size in $\mathcal{O}(n \log^*(n))$.

Theorem

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

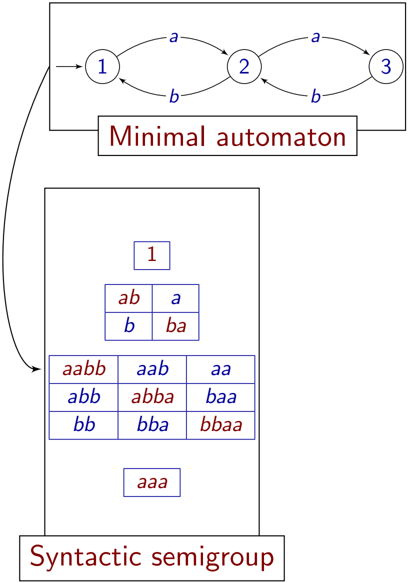
$\rightarrow$ Still open for gate linear complexity, but probably very difficult.

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



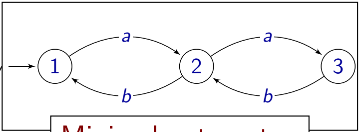
Minimal automaton

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



a
a
1
2
3
b
b
Minimal automaton
1
ab a
b ba
aabb aab aa
abb abba baa
bb bba bbaa
aaa
Syntactic semigroup

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



Minimal automaton

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

First order formula

1

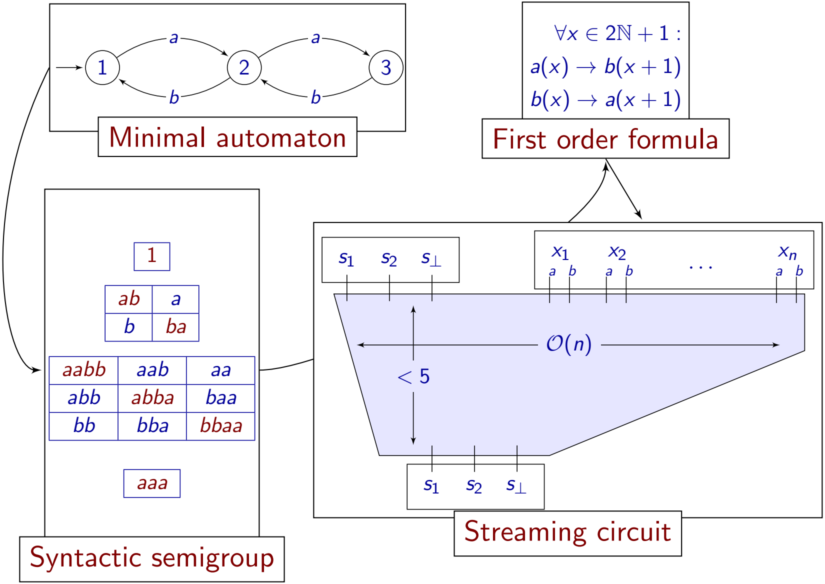
ab	a
b	ba

$aabb$	aab	aa
abb	$abba$	baa
bb	bba	$bbaa$

aaa

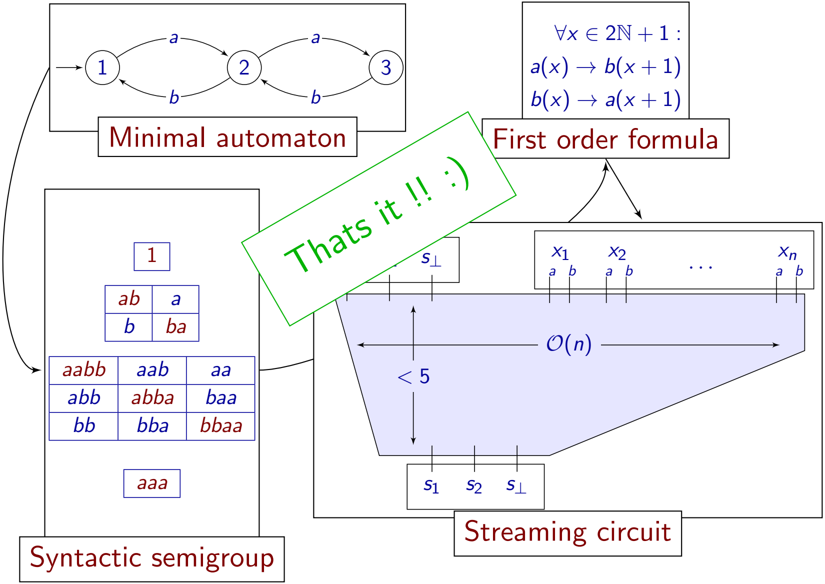
Syntactic semigroup

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



a
a
1
2
3
b
b
Minimal automaton
∀x ∈ 2ℕ + 1 :
a(x) → b(x + 1)
b(x) → a(x + 1)
First order formula
1
ab
a
b
ba
aabb
aab
aa
abb
abba
baa
bb
bba
bbaa
aaa
Syntactic semigroup
s1
s2
s⊥
x1
x2
. . .
xn
a
b
𝒪(n)
< 5
Streaming circuit

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



1
2
3
a
a
b
b
Minimal automaton
$\forall x \in 2\mathbb{N}+1:$
$a(x) \to b(x+1)$
$b(x) \to a(x+1)$
First order formula
1
ab
a
b
ba
aabb
aab
aa
abb
abba
baa
bb
bba
bbaa
aaa
Syntactic semigroup
Thats it !! :)
$s_\perp$
x_1
x_2
$\dots$
x_n
a
b
a
b
a
b
$\mathcal{O}(n)$
< 5
s_1
s_2
$s_\perp$
Streaming circuit

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^{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^{th} alternation hierarchy of first order logic with regular predicates.

$\rightarrow$ 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 ?

$\rightarrow$ Circuit complexity of unbounded regular tree languages is widely unknown.

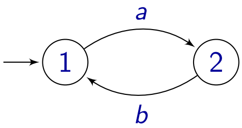
What is the block complexity of checking that an XML document satisfies a non-recursive DTD ?

$\rightarrow$ Circuit complexity of unbounded regular tree languages is widely unknown.

Theorem (Murlak, P., Pilipczuk, 2016)

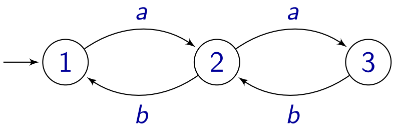
A first order definable non recursive DTD is computable by polysize and constant depth circuits.

The Dyck languages of constant depth



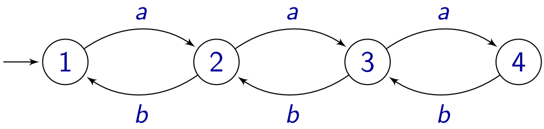
$(ab)^*$

The Dyck languages of constant depth



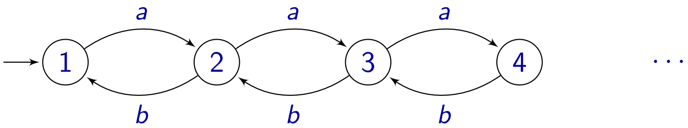
$$(a(ab)^*b)^*$$

The Dyck languages of constant depth



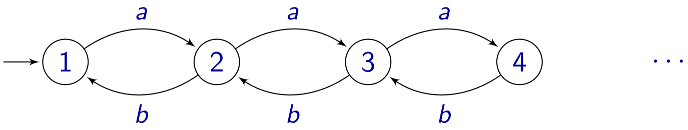
$$(a(a(ab)^*b)^*b)^*$$

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



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

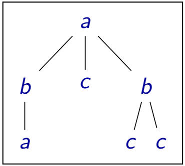
→ Each L_i is the trivial "DTD" of depth i.

→ Separate the alternation hierarchy of first order logic.

→ Probably separate the depth hierarchy as well.

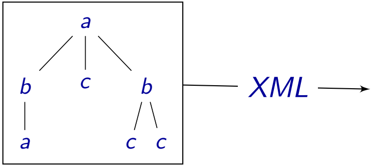
Encoding of trees

$\rightarrow$ XML documents are standard encoding of trees.



Encoding of trees

$\rightarrow$ XML documents are standard encoding of trees.



$< a >$

$< b >< a >< /a >< /b >$

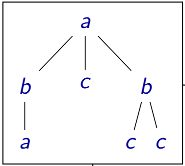
$< c >< /c >$

$< b >< c >< /c >< c >< /c >< /b >$

$< /a >$

Encoding of trees

$\rightarrow$ XML documents are standard encoding of trees.



XML

$< a >$

$< b >< a >< /a >< /b >$

$< c >< /c >$

$< b >< c >< /c >< c >< /c >< /b >$

$< /a >$

XML with depth

$< a, 0 >$

$< b, 1 >< a, 2 >< /a, 2 >< /b, 1 >$

$< c, 1 >< /c, 1 >$

$< b, 1 >< c, 2 >< /c, 2 >< c, 2 >< /c, 2 >< /b, 1 >$

$< /a, 0 >$

Simple DTD

$\rightarrow$ Simple DTD form a classical subset of DTD where productions are very constrained.

Simple DTD

$\rightarrow$ 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.