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

\[
\begin{array}{cccccccccccccccc}
  a & a & b & b & a & b & a & b & a & a & b & b & b & b & b & a & b & b & b & b & \ldots
\end{array}
\]

![Automaton Diagram]

However, data is not streamed bit by bit but by block.
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 a b b b b ...
```

```
  1  2  3
  a  a  
b  b  b
```

However, data is not streamed bit by bit but by block.
A running example: \((a(ab)^*b)^*\).
A running example: \((a(ab)^*b)^*\).

Automata: standard model for \textit{streaming} with \textit{constant} memory

\[
\begin{array}{cccccccccccccccc}
  & a & a & b & b & a & b & a & b & a & b & b & b & b & b & a & b & b & b & b & \ldots \\
\end{array}
\]

![Automata diagram](image-url)
A running example: $(a(ab)^*b)^*$. 

Automata: standard model for streaming with constant memory

```
 a  a  b  b  a  b  a  b  a  b  b  b  b  b  a  b  b  b  b  b  ... 
```

```
 1  a  2  a  3  
 b  b
```

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

\[
\begin{array}{cccccccccccccc}
a & a & b & b & a & b & a & b & a & b & b & b & b & b & b & a & b & b & b & \ldots
\end{array}
\]
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

\[
\begin{array}{cccccccccccccccc}
 a & a & b & b & a & b & a & \textcolor{green}{b} & a & a & b & b & b & b & b & a & b & b & b & \cdots \\
\end{array}
\]

\[
\begin{array}{cccccc}
 & & & & & \\
 & & & & & \\
 & & & & & \\
\end{array}
\]
A running example: \((a(ab)^*b)^*\).

Automata: standard model for \textit{streaming} with \texttt{constant} memory.

\[
\begin{array}{ccccccccccccccc}
    a & a & b & b & a & b & a & b & a & a & b & b & b & b & b & a & b & b & b & \cdots
\end{array}
\]

\[
\begin{array}{ccc}
    1 & \xrightarrow{a} & 2 \\
    b & \xrightarrow{b} & 3
\end{array}
\]
A running example: \((a(ab)^*b)^*\).

Automata: standard model for streaming with constant memory

\[
\begin{array}{cccccccccccc}
  a & a & b & b & a & b & a & b & a & a & b & b & b & b & b & b & a & b & b & b & \cdots
\end{array}
\]
A running example: \((a(ab)^*b)^*\).

Automata: standard model for **streaming** with **constant** memory

\[
\begin{array}{ccccccccccccccccccc}
  & a & a & b & b & a & b & a & b & a & a & b & b & b & b & b & a & b & b & b & \cdots \\
\end{array}
\]
A running example: \((a(ab)^*b)^*\).

Automata: standard model for **streaming** with **constant** memory

\[
\begin{array}{cccccccccccccccc}
a & a & b & b & a & b & a & b & a & a & b & b & b & b & a & b & b & b & b & \cdots
\end{array}
\]
A running example: \((a(ab)^*b)^*\).

Automata: standard model for **streaming** with **constant** memory

- Diagram:
  - States: 1, 2, 3
  - Transitions: 1→2 (a, b), 2→3 (a), 3→1 (b)

- Sequence:
  - \(a\ a\ b\ b\ a\ b\ a\ b\ a\ a\ b\ b\ b\ b\ b\ b\ a\ b\ b\ b\ b\ b\ b\ \cdots\)
A running example: \((a(ab)^*b)^*\).

Automata: standard model for streaming with constant memory

\[
\begin{array}{cccccccccccc}
a & a & b & b & a & b & a & b & a & b & b & b & b & b & a & b & b & b & b & \cdots
\end{array}
\]

![Automata Diagram]

However, data is not streamed bit by bit but by block.
A running example: \((a(ab)^*b)^*\).

Automata: standard model for \textit{streaming} with \textit{constant} memory

\[
\begin{array}{cccccccccccccccc}
\text{a} & \text{a} & \text{b} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} & \text{b} & \text{b} & \text{b} & \cdots
\end{array}
\]
A running example: \((a(ab)^*b)^*\).

Automata: standard model for **streaming** with **constant** memory

\[
\begin{array}{c}
a & a & b & b & a & b & a & b & a & b & b & b & b & b & a & b & b & b & b & \cdots
\end{array}
\]
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  b  ...  
```

![Diagram of an automaton with states 1, 2, and 3, transitions labeled with symbols a and b.](image)
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

\[
\begin{array}{ccccccccccccccccc}
a & a & b & b & a & b & a & b & a & b & b & b & b & b & a & b & b & b & \cdots \\
\end{array}
\]
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

\[
\begin{array}{cccccccccccc}
  a & a & b & b & a & b & a & b & a & b & b & b & b & b & a & b & b & b & \cdots
\end{array}
\]

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 running example: \((a(ab)^*b)^*\).

\[a\ a\ b\ b\ a\ b\ a\ b\ a\ a\ a\ b\ a\ b\ b\ b\ a\ a\ a\ \cdots\]

\[\begin{array}{cc}
1 & \rightarrow \text{many} \\
2 & \text{too much} \\
\end{array}\]

\[\begin{array}{cc}
1 & \rightarrow \text{a lot} \\
2 & \cdots \\
\end{array}\]

\[\rightarrow \text{Number of transitions grows exponentially with the block size.}\]
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

\[ \begin{align*}
\neg a \land b \\
\neg \land a \land b \\
\neg b \\
\neg \land b \\
a \land b \\
\neg a \land b
\end{align*} \]

Recognizes \((a(ab) \ast b) \ast \cap A_4 = \{abab, aabb\}\).
Circuits

\[ \overline{\overline{a \overline{b} \overline{a} b} \overline{a \overline{b} \overline{a} b}} = \{abab, aabb\} \]
Circuits

\[ \text{Recognizes } (a (ab)*) * \cap A_4 = \{abab, aabb\} \]
Circuits

Recognizes \((a(ab)) \ast b \ast \cap A_4 = \{abab, aabb\}\).
Circuits

\[ \text{Recognizes } (a \text{ and } b) \star b = \{abab, aabb\}. \]
Circuits

\[ \rightarrow \text{Recognizes } (a(ab)^* b)^* \cap A^4 = \{abab, aabb\}. \]
Circuits family

→ Recognizes $(a(ab)^* b)^* \cap A^6$. 
→ Recognizes \((a(ab)^* b)^* \cap A^{10}\).
Circuits family

→ Recognizes \((a(ab)^*b)^* \cap A^{14}\).
→ Recognizes $a(ab)^*b^* \cap A^{2n}$ with const. depth and linear size.

→ Non uniform model of computation.
The streaming circuits
The streaming circuits

![Diagram showing the streaming circuits with states 1 and 2, transitions ab, aa, and ba, and a block with variables x1, x2, a, b, s1, s2, and s⊥. The circuit diagrams and logic gates represent the flow of inputs and outputs.]
The streaming circuits

states in
1 0 0

block
a a
1 0 1 0

states out
s_1 s_2 s_⊥
The streaming circuits

\[ ab \quad aa \quad ba \]
\[ bb \]

\[ 1 \quad 2 \]

states in
\[ 1 \quad 0 \quad 0 \]

block
\[ a \quad a \]
\[ 1 \quad 0 \quad 1 \quad 0 \]

\[ \land \quad \land \quad \land \quad \land \quad \land \quad \land \quad \land \quad \land \quad \lor \quad \lor \quad \lor \quad \lor \quad \lor \quad \lor \quad \lor \quad \lor \quad \lor \quad \lor \quad \lor \quad \land \quad s_1 \quad s_2 \quad s_\perp \]

states out
The streaming circuits
The streaming circuits
The streaming circuits

![Diagram of streaming circuits with states and transitions labeled as follows: states in 1 0 0, block 1 0 1 0, and states out s1 s2 s⊥.](image-url)
The streaming circuits

![Diagram of streaming circuits](image)
The streaming circuits
The streaming circuits

![Diagram of streaming circuits](image_url)
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\(^1\)-complexity**

**Proposition (Folklore).**

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

→ A simple divide and conquer.
→ Provides already nice hardware synthesis.
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$^1$-complexity**

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

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

**Can we do better?**

**Theorem (Barrington, 1989).**
Regular languages are complete for $\text{NC}^1$. 
$\textbf{NC}^1$-complexity

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

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

Can we do better? *No! :(*

**Theorem (Barrington, 1989).**
Regular languages are complete for $\textbf{NC}^1$. 
Back to \((a(ab)^*b)^*\)

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

When can we improve the \(\textbf{NC}^1\) construction?
Theorem (Barrington et al., 1992).

Regular languages computable with const depth and polysize are exactly those definable in first order logic with regular predicates.
**AC⁰**-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 \texttt{const} depth and \texttt{polysize} are exactly those definable in \texttt{first order} logic with \texttt{regular predicates}. Moreover this class has a \texttt{decidable} membership.

→ it is a consequence of two importants results:

1. Parity is not in **AC^0**.
   
   Furst, Saxe and Sipser (1984)

2. For regular languages, \texttt{first-order} definability is equivalent to \texttt{aperiodicity}.
   
   Schützenberger (1965), McNaughton and Papert (1971).
Back to \((a(ab)^*b)^*\)

![Minimal automaton diagram](image-url)
Back to \((a(ab)^*b)^*\)

Minimal automaton

Syntactic semigroup
Back to \((a(ab)^*b)^*\)

**Minimal automaton**

1. \(a\) from 1 to 2
2. \(b\) from 2 to 3
3. \(a\) from 3 to 1

**Syntactic semigroup**

<table>
<thead>
<tr>
<th></th>
<th>(ab)</th>
<th>(a)</th>
<th>(b)</th>
<th>(ba)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(aabb)</td>
<td>(aab)</td>
<td>(aa)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(abb)</td>
<td>(abba)</td>
<td>(baa)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(bb)</td>
<td>(bba)</td>
<td>(bbaa)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(aaa)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**First order formula**

\[\forall x a(x) \land a(x+1) \rightarrow \exists y b(y) \land b(y+1) \land \forall z \in [x, y] \land a(z) \leftrightarrow b(z+1)\]
Back to \((a(ab)^* b)^*\)

Minimal automaton

\[
\begin{align*}
1 & \xrightarrow{a} 2 \xrightarrow{a} 3 \\
& \xrightarrow{b} 2 \xrightarrow{b} 3
\end{align*}
\]

Syntactic semigroup

\[
\begin{array}{ccc}
ab & a & b \\
ba & \text{aaa} & \text{aabb} \\
abb & \text{aab} & \text{aa} \\
bb & \text{abba} & \text{baa} \\
& \text{bba} & \text{bbaa}
\end{array}
\]

First order formula

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

Streaming circuit

\[
\begin{array}{cccc}
s_1 & s_2 & s_\bot & x_1 \\
a & b & a & b \\
s_1 & s_2 & s_\bot & x_2 \\
a & b & a & b \\
\vdots & \vdots & \vdots & \vdots \\
s_1 & s_2 & s_\bot & x_n \\
a & b & a & b
\end{array}
\]

\[\mathcal{O}(n^3)\]
Back to \((a(ab)^*b)^*\)

Minimal automaton

\[
\begin{array}{ccc}
1 & \xrightarrow{a} & 2 \\
& \xleftarrow{b} & \\
2 & \xrightarrow{a} & 3 \\
& \xleftarrow{b} & \\
\end{array}
\]

First order formula

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

Syntactic semigroup

\[
\begin{array}{c|c|c}
ab & a & \\
\hline
b & ba & \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c}
aabb & aab & aa & \\
\hline
abb & abba & baa & \\
\hline
bb & bba & bbba & \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c}
aaa & \\
\hline
\end{array}
\]

Streaming circuit

\[
O(n^3) < 6
\]

Not the best!!
Size and depth tradeoff

The CFL procedure

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

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

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 $\text{AC}^0$ are computable by circuits with wire-size in $O(n \log^*(n))$.

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 $\text{AC}^0$ are computable by circuits with wire-size in $O(n \log^*(n))$.

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

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 \((a(ab)^*b)^*\)

Minimal automaton
Back to \((a(ab)^*b)^*\)
Back to \((a(ab)^*b)^*\)

---

### Minimal automaton

1. \(a\)
2. \(b\)
3. \(a\)

---

### Syntactic semigroup

<table>
<thead>
<tr>
<th></th>
<th>(ab)</th>
<th>(a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>(ba)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(aabb)</th>
<th>(aab)</th>
<th>(aa)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(abb)</td>
<td>(abba)</td>
<td>(baa)</td>
<td></td>
</tr>
<tr>
<td>(bb)</td>
<td>(bba)</td>
<td>(bbaa)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(aaa)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### First order formula

\(\forall x \in 2\mathbb{N} + 1 :\)

\(a(x) \rightarrow b(x + 1)\)

\(b(x) \rightarrow a(x + 1)\)
Back to \((a(ab)^*b)^*\)

**Minimal automaton**

\[
\begin{array}{ccc}
1 & \xrightarrow{a} & 2 \\
& \xrightarrow{b} & 3
\end{array}
\]

**Syntactic semigroup**

\[
\begin{array}{|c|c|}
\hline
ab & a \\
\hline
b & ba \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
aabb & aab & aa \\
abb & abba & baa \\
bb & bba & bbba \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
aaa & \\
\hline
\end{array}
\]

**First order formula**

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

**Streaming circuit**

\[
\begin{array}{|c|c|c|}
\hline
s_1 & s_2 & s_\perp \\
\hline
x_1 & a & b \\
x_2 & a & b \\
\vdots & \vdots & \vdots \\
x_n & a & b \\
\hline
\end{array}
\]

\[O(n) < 5\]
Back to \((a(ab)^*b)^*\)

Minimal automaton

First order formula

\[ \forall x \in 2\mathbb{N} + 1 : \\
\begin{align*}
a(x) &\rightarrow b(x + 1) \\
b(x) &\rightarrow a(x + 1)
\end{align*} \]

Syntactic semigroup

Streaming circuit

That's it!! :)

\[
\begin{array}{c|c|c}
1 & 2 & 3 \\
\hline
a & b & a \\
\hline
b & ba
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{ab} & \text{a} & \text{aabb} \\
\hline
\text{b} & \text{ba} & \text{abb} \\
\hline
\text{aab} & \text{abba} & \text{abb} \\
\text{a} & \text{baa} & \text{baa} \\
\text{b} & \text{bb} & \text{bb} \\
\text{b} & \text{baa} & \text{bb} \\
\text{b} & \text{baa} & \text{bb} \\
\text{a} & \text{aaa} & \text{aaa}
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\text{x}_1 & \text{x}_2 & \ldots & \text{x}_n \\
\hline
\text{a} & \text{b} & \text{a} & \text{b} \\
\hline
\text{a} & \text{b} & \text{a} & \text{b} \\
\hline
\text{a} & \text{b} & \text{a} & \text{b} \\
\hline
\text{a} & \text{b} & \text{a} & \text{b} \\
\hline
\text{a} & \text{b} & \text{a} & \text{b} \\
\hline
\text{a} & \text{b} & \text{a} & \text{b} \\
\hline
\text{a} & \text{b} & \text{a} & \text{b}
\end{array}
\]

\[ O(n) \]

\[ < 5 \]
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.
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 non-recursive 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

\[
\begin{align*}
L_0 &= \epsilon \\
L_{i+1} &= (aL_ib)^*
\end{align*}
\]

Separate the alternation hierarchy of first order logic.

Separate the depth hierarchy as well.
The Dyck languages of constant depth

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.

\[
\begin{align*}
L_0 &= \epsilon \\
L_{i+1} &= (aL_i b)^* 
\end{align*}
\]
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.