Schema validation via streaming circuits[*]

Filip Murlak
University of Warsaw
fmurlak@mimuw.edu.pl

Charles Paperman
University of Warsaw
paperman@mimuw.edu.pl

Michał Pilipczuk
University of Warsaw
michal.pilipczuk@mimuw.edu.pl

ABSTRACT

XML schema validation can be performed in constant memory in the streaming model if and only if the schema admits only trees of bounded depth—a mild assumption from the practical view-point. In this paper we refine this analysis by taking into account that data can be streamed block-by-block, rather then letter-by-letter, which provides opportunities to speed up the computation by parallelizing the processing of each block. For this purpose we introduce the model of streaming circuits, which process words of arbitrary length in blocks of fixed size, passing constant amount of information between blocks. This model allows us to transfer fundamental results about the circuit complexity of regular languages to the setting of streaming schema validation, which leads to effective constructions of streaming circuits of depth logarithmic in the block size, or even constant under certain assumptions on the input schema. For nested-relational DTDs, a practically motivated class of bounded-depth XML schemas, we provide an efficient construction yielding constant-depth streaming circuits with particularly good parameters.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages

General Terms

Theory, Algorithms

Keywords

semi-structured data, XML, streaming, schema validation, Boolean circuits, nested-relational DTDs

1. INTRODUCTION

Over tree-structured data, like XML documents or JSON files, schemas impose restrictions on the structure of the trees modelling the data. Popular schema formalisms, like DTDs and especially XML Schema, are able to express very complex properties, bringing their expressive power close to tree automata, which are often used as theoretical abstractions of schemas. With such expressive power, the task of schema validation—that is, verifying that a given data instance conforms to the schema—is not entirely trivial even if we have direct access to the whole data instance. When the data are streamed, schema validation becomes a major challenge.

In their seminal paper [28], Segoufin and Vianu consider streaming validation in constant memory. An algorithm over streamed data that works in constant memory can be seen as a finite automaton. Whether such an algorithm exists for a given schema depends on whether the set of word representations of the instances of the schema is a regular language. Segoufin and Vianu show that the word representation of a regular tree language (covering all popular schema formalisms) is regular if and only if there exists a uniform bound on the depth of the trees in the language. In this paper we refine this result by looking more closely at the way the data are streamed. Due to the result of Segoufin and Vianu, we focus predominantly on tree languages of bounded depth.

Our starting point is the observation that data need not be fed to the algorithm letter-by-letter. For instance, if the data stream serves as an abstraction of sequential access to a mass storage device, the algorithm is fed entire blocks of data that are fetched to a moderately-sized cache. This requires evaluating the finite automaton over a word read in block-sized portions. It can still be done letter-by-letter, giving time linear in the size of the block, but we would like to do better, assuming certain ability to parallelize computation. As a model of parallelism we choose Boolean circuits. The most important reason is that their relation with regular languages is well understood and documented [1, 17, 18, 29], but Boolean circuits have several other advantages. On one hand, they are very close to hardware implementation: from a Boolean circuit of small depth one can directly obtain a hardware description that could be compiled into, say, an FPGA. On the other hand, Boolean circuits are also a commonly accepted theoretical model for higher-level parallelism, providing abstraction for various concrete practical models. For example, on a multi-core machine different cores could be assigned to evaluating different

[*]These results were obtained when Charles Paperman and Michał Pilipczuk held post-doc positions at the Warsaw Centre of Mathematics and Computer Science. Michał Pilipczuk was supported by the Foundation for Polish Science (FNP) via the START stipend programme. Filip Murlak was supported by Poland's National Science Centre grant no. UMO-2013/11/D/ST6/03075.

parts of the circuit. We remark that combining the challenges of streaming data access and parallelization has been considered from the practical perspective [3, 4, 10, 15], in particular in the context of the standard MapReduce approach (see e.g. [21]); however, to the best of our knowledge, hardly any theoretical models have been proposed so far.

In order to reconcile the random-access parallelism of Boolean circuits with the streaming setting, we introduce a model of computation called streaming circuits. Intuitively, a streaming circuit takes a block of the input word together with additional feedback information of constant size (the state of the underlying finite automaton) and outputs updated feedback. This model allows us to talk about the complexity of streaming algorithms in a way that does not abstract away the size of the block, and to transfer the huge body of results on the circuit complexity of regular languages to the streaming setting. It also avoids the inherent flaw of the classical Boolean circuit setting: the nonuniformity. While having a separate circuit for each size of the input data is entirely impractical, in our setting this is not an issue any more, as the circuit is nonuniform in the block size, which can be chosen and fixed in advance.

Any finite automaton can be transformed into a streaming circuit of chosen block size. The challenge is to get the circuit as simple as possible. In the context of schema validation we are interested in how the complexity of tree language is reflected in the streaming circuits recognizing their word encodings. As we shall see, one can always build an NC^1-streaming circuit for any bounded-depth regular tree language. That is, in the block-by-block access model, one can efficiently do streaming validation in parallel—by a circuit that has polynomial size, constant fan-in, and logarithmic depth. Can we do better than that? For any class $\mathcal{C}$ of circuits one can ask about streaming $\mathcal{C}$-circuits for regular tree languages. A full positive answer to this question should include:

- an algorithm do decide if a given tree language has a streaming $\mathcal{C}$-circuit;
- an algorithm to construct a recognizing $\mathcal{C}$ streaming circuit, if it exists; and
- a syntactic fragment (a restricted schema language) that corresponds to languages that can be recognized with a $\mathcal{C}$ streaming circuit.

From the practical point of view, the crucial part of the answer is a restricted schema language guaranteeing feasibility and an efficient algorithm to build the circuit from the schema definition in this restricted language. Ideally, the schema language should cover all feasible schemas, but this should not be achieved at the expense of its simplicity and usability.

We consider two restricted classes of circuits: AC^0 and WLAC^0. Recall that AC^0 comprises circuit families with polynomial size and constant depth bounds, and an AC^0 circuit family is in WLAC^0 (for *wire-linear* AC^0) if the number of wires is bounded linearly. Unlike for NC^1, not all bounded-depth regular tree languages admit AC^0 streaming circuits, but we can decide effectively if a given bounded-depth regular language admits one. We also show that if one additionally assumes that the tree language is definable in first order logic, then the answer is always affirmative. For a practically relevant class of languages defined by nested-related DTDs [2], we provide an efficient construction of AC^0 streaming circuits with particularly good properties. For WLAC^0 we observe that one cannot even check the correctness of the usual XML encodings of bounded-depth trees. We propose a new encoding, enriched with the information about ancestors of nodes. Under the new encoding we can validate nested-relational schemas with WLAC^0 streaming circuits. We also show that this encoding can be computed from the usual encoding by an AC^0 streaming circuit that does not depend on the schema, only on the depth and the alphabet.

2. STREAMING CIRCUITS

In this article we use several classical classes of circuits. We briefly recall basic definitions and refer to the book of Straubing [29] for a more detailed presentation.

Basics of circuit complexity. We work with Boolean circuits with AND, OR, and NOT gates, taking as input words over alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$. The letters of the input word are encoded in unary; that is, each input gate is modelled with k binary gates, and letter a_i is encoded as the binary sequence $0^{i-1}10^{k-i}$. A family

$$(C_n)_{n\in\mathbb{N}}$$

of circuits recognizes a language $L \subseteq \Sigma^*$ if C_n has n input gates and a single binary output gate, and returns 1 if and only if the input word is in L. We shall refer to this model of recognition as the random-access model. Whenever we consider the size of the circuit, we mean the number of gates.

Since we consider mainly regular languages (of words and of trees), we restrict ourselves to languages recognizable with NC^1 circuit families: Boolean circuit families of polynomial size, logarithmic depth and bounded fan-in. Two other classes of interest are AC^0 circuit families, which have polynomial size, constant depth and unbounded fan-in, and WLAC^0 circuit families, which are AC^0 circuit families with a linear number of wires (hence, also gates). The interaction of these classes with the class of regular languages is well understood, and we will use this knowledge to design adequate devices in the context of streaming schema validation.

We shall say that a language is in class $\mathcal{C}$ if it is recognized by a $\mathcal{C}$ circuit family. Some important examples separating the three classes described above:

- the parity language $(c + ac^*a)^*$ is in NC^1 (with linear-size circuits) but not in AC^0 [12];
- $(c + ac^*b)^*$ is in AC^0 (with quadratic-size circuits) but not in WLAC^0 [17].

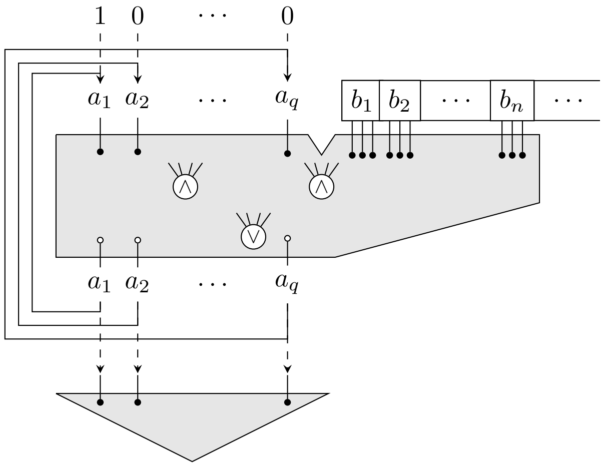
Streaming circuits. In the streaming circuit setting, a single circuit is used to recognize words of all lengths, by processing them sequentially in blocks of fixed size with the help of a feedback mechanism.

A *streaming circuit* over alphabet Σ with block size n and feedback size m is a circuit C with n input gates, m feedback gates and m output gates, together with an acceptor circuit A with m input gates and 1 output gate. The computation of such a circuit on an input word w is carried out in stages. In each stage the circuit C is given the output from the previous stage (initially, the bit sequence 10^{m-1}) and the next size n block of the input word (in unary encoding). The output of the last stage is fed to the acceptor circuit A and the word w is *accepted* if and only if the acceptor circuit A returns 1. If the last block of the input word is shorter than n symbols, a designated padding symbol \$ (encoded as a sequence of zeros) is used to fill it up. More formally, let $u_0 = 10^{m-1}$ and for $i < \left\lceil \frac{|w|}{n} \right\rceil$ let

$$u_{i+1} = C(u_i, w_{ni+1}w_{ni+2} \dots w_{ni+n})$$

where $w_j = \$$ for $j > |w|$; the word w is accepted if

$$A(u_{\lceil \frac{|w|}{n} \rceil}) = 1\,.$$



Such a streaming circuit can be interpreted as a deterministic automaton over the alphabet $\Gamma = \Sigma^n$: the state space is $\{0,1\}^m$ with the initial state 10^{m-1}, and the transition function and the set of accepting states are given by circuits C and A. Consequently, languages recognized by streaming circuits are regular.

In fact, streaming circuits give precise description of the implementation of finite automata over words read by fixed-size blocks. Indeed, if you take an automaton and make it read blocks of n letters instead of one letter, you obtain an automaton with the same state space over the alphabet Σ^n, but with the set of transitions growing exponentially with n (if $|\Sigma| \geq 2$). Circuits allow us to represent (and carry out) transitions more efficiently.

We can therefore talk about streaming-circuit complexity of regular languages: a regular language L has *streaming-circuit complexity* $\mathcal{C}$ if for some m there exists a $\mathcal{C}$ circuit family $(C_n)_{n\in\mathbb{N}}$ with m feedback gates and m output gates and an acceptor circuit A with m input gates, such that for each n, the streaming circuit (C_n, A) recognizes L. We say that $(C_n)_{n\in\mathbb{N}}$ is a $\mathcal{C}$ *streaming circuit family* for L (with feedback m and acceptor A). In general, $(C_n)_{n\in\mathbb{N}}$ need not have a finite description, but all families constructed in this paper will have one.

Streaming vs random access. Over regular languages, random-access recognizability and streaming recognizability coincide for reasonable classes of circuits. The following theorem provides efficient translation from circuits to streaming circuits, and *vice versa.* A class $\mathcal{C}$ of circuits families is *closed under shifts* if for each circuit family $(C_n)_{n\in\mathbb{N}}$ from $\mathcal{C}$, each family obtained from $(C_{n+1})_{n\in\mathbb{N}}$ by hard-wiring a chosen input gate is in $\mathcal{C}$.

THEOREM 1. *Let $\mathcal{C}$ be a class of circuit families closed under shifts and Boolean combinations. Then a regular language $L \subseteq \Sigma^*$ has streaming-circuit complexity $\mathcal{C}$ if and only if it is in $\mathcal{C}$.*

More precisely, if the recognizing family of circuits has depth and size bounded by non-decreasing functions $d(n)$ and $s(n)$, the resulting streaming circuit for block size n has feedback k, depth $d(n+k+k^2)+\mathcal{O}(1)$, and size $k^3 \cdot s(n+k+k^2)+\mathcal{O}(k^3)$, where k is the number of states of the minimal deterministic automaton for L.

PROOF. The left to right implication is almost immediate. Let $(C_n)_{n\in\mathbb{N}}$ be a $\mathcal{C}$ streaming circuit family for L with feedback m and acceptor A. The family of circuits recognizing L is

$$\big(A(C_n(10^{m-1}, \cdot))\big)_{n\in\mathbb{N}}\,;$$

that is, we hardwire the initial values in the feedback gates of C_n, and feed the output of C_n to A. Since circuit A is fixed, by the closure properties of $\mathcal{C}$, the resulting circuit family is indeed in $\mathcal{C}$, and so is L.

The right to left implication is more complicated. Let $\mathcal{A}$ be the minimal deterministic automaton for L and $(C_n)_{n\in\mathbb{N}}$ a $\mathcal{C}$ family of circuits recognizing L. Let p, q be states of $\mathcal{A}$. We shall construct a $\mathcal{C}$ family of circuits recognizing the language

$$L_{p,q} = \{w \in \Sigma^* \mid \delta_w(p) = q\}\,,$$

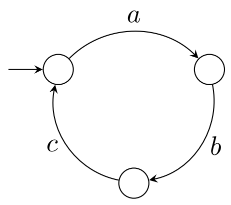
where $\delta_w(p)$ is the state to which the automaton moves from state p after reading word w. Let u be the shortest word that reaches p from the initial state; by pumping, $|u_p| \leq k$, where k is the number of states of $\mathcal{A}$. By minimality, for all distinct states q, q' there is a word v of size at most k^2 such that $\delta_v(q) \in F$ if and only if $\delta_v(q') \notin F$. Consequently, $L_{p,q}$ is a Boolean combination of $k-1$ *residual* languages of the form

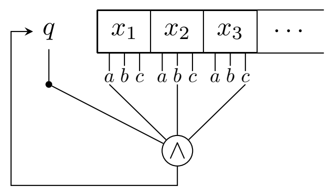
$$u^{-1}L\,v^{-1} = \{w \mid uwv \in L\}\,,$$

where $|u| \leq k$ and $|v| \leq k^2$. Each of these languages can be recognized with a $\mathcal{C}$ family of circuits $(C'_n)_{n\in\mathbb{N}}$ where C'_n is obtained from $C_{n+|u|+|v|}$ by hardwiring the first $|u|$ input gates to u and the last $|v|$ input gates to v. An appropriate Boolean combination of these circuits gives circuits for $L_{p,q}$. Assuming that i-th state is coded as $0^{i-1}10^{k-i}$ on the feedback gates and the first state is initial, it is easy to obtain a $\mathcal{C}$ family of streaming circuits for L from the circuits for languages $L_{p,q}$. We simply add on top of these circuits an additional circuit of depth $\mathcal{O}(1)$ and size $\mathcal{O}(k^2)$ that computes the state after processing the current block from the previous state passed in the feedback. To verify the required size bound, observe that in total we construct k^2 circuits for languages $L_{p,q}$. Every such circuit consists of $k-1$ circuits for residual languages, each of size at most $s(n+k+k^2)$. Since k is considered a constant, by the closure properties of $\mathcal{C}$ we have that the obtained circuit family belongs to $\mathcal{C}$. $\square$

One could easily make the feedback logarithmic in k by encoding the state in binary. However, this would not improve the parameters of the circuit, as they depend on the number of states, not the size of their representation.

The fact that the circuit has access, thanks to its non-uniformity, to additional numerical information, sometimes allows to simplify drastically the ongoing computation. For instance, the language $(a_1a_2\cdots a_n)^*$ requires an automaton with $n+1$ states, but assuming block size n it can be recognized by a streaming circuit with feedback 1, which corresponds to 2 states:





This kind of behaviour is difficult to analyze, but always beneficial in the construction of streaming circuits.

3. VALIDATION: A GENERAL BOUND

Following Segoufin and Vianu [28], we work with ordered unranked trees, node-labelled with letters from a finite alphabet Σ. We denote by Trees(Σ) the set of all such trees.

We model schemas as "previous sibling, last child" tree automata. A *nondeterministic tree automaton*

$$\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$$

consists of a finite input alphabet Σ, a finite set of states Q with an initial state q_0, a set of accepting states $F \subseteq Q$, and a transition relation

$$\delta \subseteq Q \times Q \times \Gamma \times Q\,.$$

Being in a node v of the input tree $t \in \text{Trees}(\Sigma)$, the automaton has processed t_v, the subtree of t rooted at v. The state q for node v depends on the label σ of v and the states q_1, q_2 from the previous sibling and the last child of v, respectively, in the way specified by the transition relation:

$$(q_1, q_2, \sigma, q) \in \delta$$

(in leftmost siblings and leaves we use the initial state q_0 instead of q_1 and q_2, respectively). The tree t is *accepted* by $\mathcal{A}$ if states can be chosen for nodes in such a way that the root gets a state from F. We write $L(\mathcal{A})$ for the set of accepted trees. If $L = L(\mathcal{A})$, we say that L is *regular*, and that it is recognized by $\mathcal{A}$.

A schema language that is simpler, but often sufficient in practice, is offered by *document type definitions*, or DTDs for short. A DTD

$$\mathcal{D} = (\Sigma, r, P)$$

consists of a finite alphabet Σ with a distinguished root label $r \in \Sigma$, and a function P that assigns to each label $a \in \Sigma$ a regular expression $P(a)$ over Σ, called the production for a, and written as $a \to P(a)$. A tree $t \in \text{Trees}(\Sigma)$ conforms to $\mathcal{D}$ if its root is labelled with r and for each label $a \in \Sigma$ and each node v in t with label a, the sequence of labels of v's children forms a word generated by the regular expression $P(a)$.

A practically relevant class of *nested-relational* DTDs, covering a large proportion of real life schemas [2], is obtained by assuming non-recursiveness (that is, no a-labelled node has an a-labelled descendant) and allowing only productions of the form

$$a \to \widehat{a}_1\widehat{a}_2\ldots\widehat{a}_\ell\,,$$

where $a_1, a_2, \ldots a_\ell$ are distinct elements of Σ, and $\widehat{a}_i$ is equal to a_i, $a_i? = (\varepsilon + a_i)$, a_i^*, or $a_i^+ = a_i a_i^*$.

In the context of streaming processing we need a string representation of trees. Under the *XML encoding* trees are represented as words over $\Sigma \cup \overline{\Sigma}$, the elements of Σ and $\overline{\Sigma}$ being, respectively, the *opening* and *closing tags*,

$$\text{flat}\left(\overset{a}{t_1 \ \cdots \ t_k} \right) = a \cdot \text{flat}(t_1)\cdots\text{flat}(t_k)\cdot\overline{a}\,.$$

We call flat(t) the *flattening of* t, and

$$\text{flat}(L) = \big\{\text{flat}(t) \bigm| t \in L\big\}$$

the *flattening of* L. Another natural possibility is the *term encoding*, which is similar to the XML encoding except that we only have one closing tag symbol #,

$$\text{flat}_\#\left(\overset{a}{t_1 \ \cdots \ t_k} \right) = a \cdot \text{flat}_\#(t_1)\cdots\text{flat}_\#(t_k)\cdot \#\,.$$

Intuitively, the XML encoding corresponds to recognition by a *visibly push-down automaton* [20] (or input driven automaton [11]). The term encoding requires only one stack symbol, which corresponds visibly counter automata. In the sequel we work with the XML encoding for concreteness, but for most of our results, the choice of the encoding does not matter.

As observed by Segoufin and Vianu [28], the flattening of a regular tree language is a regular word language if and only if the tree language has bounded depth (there exists a uniform bound on the depth of all trees in L).

PROPOSITION 1 ([28]). *For each regular tree language L, the following conditions are equivalent:*

- *L has bounded depth;*
- *flat(L) is a regular word language.*

Thus, to have any chance for streaming-circuit validation, we restrict our attention to bounded-depth trees.

From the practical point of view this is a mild assumption, as real-life schemas tend to be bounded-depth [2].

Translation from bounded-depth tree automata to *deterministic* word automata over encodings involves only single-exponential blow-up.

Proposition 2. *Let $\mathcal{A}$ be a tree automaton with k states recognizing a bounded-depth language $L \subseteq \mathrm{Trees}(\Sigma)$. One can construct a deterministic automaton with $\mathcal{O}(|\Sigma|^k \cdot 2^{k^2})$ states recognizing* $\mathrm{flat}(L)$. (♠) [1]

We finish this section with a general NC^1-upper bound for streaming circuit complexity of bounded-depth regular tree languages. Assuming the usual interpretation of NC^1 as the class of problems that can be solved efficiently in parallel, this shows that streaming validation parallelizes. The bound combines Theorem 1, Proposition 2 and a folklore fact that all regular languages are in NC^1 (i.e., random-access validation parallelizes).

Proposition 3. *Each regular language L can be recognized by an NC^1 streaming circuit family. More precisely, for block size n one can construct a recognizing circuit with feedback k, depth $\mathcal{O}(\log n)$, and size $\mathcal{O}(k^3 n)$, where k is the number of states of the minimal deterministic automaton for L.*

Proof. The first part of the claim follows by Theorem 1 from the fact that all regular languages are in NC^1. To achieve the claimed bounds, we adapt the standard construction to obtain directly a streaming circuit.

Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be the minimal automaton for L, and let $|Q| = k$. Any function $f\colon Q \to Q$ can be represented as a $k \times k$ binary matrix, which in turn can be seen as a k^2-bit word. From a single input letter σ one can compute function δ_σ, represented as a k^2-bit word, using a circuit that has depth 1 and size $\mathcal{O}(k^2)$; note that this circuit hardwires the transition function of $\mathcal{A}$. Given two $k \times k$ matrices (as k^2-bit words), one can compute their product (over the Boolean algebra) with a circuit of depth 2 and size $\mathcal{O}(k^3)$. Finally, for an input word w of length $n = 2^k$ one can compute the function δ_w by first computing functions δ_{w_i}, and then composing them in pairs to obtain functions for pairs of consecutive letters, quadruples, octuples, etc. The resulting circuit C_n has depth $\mathcal{O}(\log n)$ and size $\mathcal{O}(k^3 n)$. If $|w|$ is not a power of two, take $C_{2^{\lceil \log n \rceil}}$ and hardwire identity function in place of the functions for last $2^{\lceil \log n \rceil} - |w|$ input letters; this preserves the bounds. From the family $(C_n)_{n \in \mathbb{N}}$ one easily constructs a streaming circuit family for L: one simply computes the value of the function computed by C_n on the state represented (in unary) by the values in the feedback gates. This can be done easily with a circuit of depth 2 and size $\mathcal{O}(k^2)$. □

Propositions 2 and 3 immediately yield the following.

Theorem 2. *For each regular bounded-depth language $L \subseteq \mathrm{Trees}(\Sigma)$, $\mathrm{flat}(L)$ can be recognized by an NC^1 streaming circuit family. More precisely, for block size n, one can construct a recognizing streaming circuit with feedback $\mathcal{O}(|\Sigma|^k \cdot 2^{k^2})$, depth $\mathcal{O}(\log n)$, and size $\mathcal{O}(|\Sigma|^{3k} \cdot 2^{3k^2} \cdot n)$, where k is the number of states of the given nondeterministic automaton recognizing L.* □

As remarked in [28], if the input tree language is given as a DTD with productions defined by *nonambiguous* regular expressions, one can construct a finite automaton recognizing either encoding, with the number of states bounded by a polynomial of degree at most $|\Sigma|$. This immediately improves the bound in the theorem above.

As we have seen in Theorem 1, for regular word languages streaming and random-access recognition with circuits is the same for any reasonable class of circuits. For flattenings of regular tree languages, this is not the case. In the streaming model, the flattening must be regular to be recognized by any circuit family; in the random-access model, the flattening of each regular tree language can be recognized by an NC^1 circuit family [11].

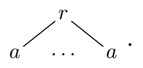
4. VALIDATION IN CONSTANT DEPTH

In the last section we saw a generic construction translating a description of a tree language (an automaton or a DTD) into a streaming circuit with relatively good parameters: logarithmic depth, constant fan-in, and polynomial size. However, this construction is largely suboptimal as shown by the following example.

Example 1. *Let L be given by the following DTD*

$$r \to a^* \,;$$

that is, L consists of trees of the form



Then, $\mathrm{flat}(L) = r(a\overline{a})^*\overline{r}$ can be easily recognized by an AC^0 streaming circuit family (of linear size).

On the other hand, any regular language not in AC^0 gives a depth-2 regular tree language whose flattening is not recognizable by an AC^0 streaming circuit family.

Example 2. *From the parity lower bound [12] we immediately get that for the tree language L given by*

$$r \to (ab^*a + b)^* \,,$$

$\mathrm{flat}(L)$ cannot be recognized by an AC^0 family of streaming circuits.

Yet again, a simple modification can turn a hard tree language into an easy one, by adding more structure.

Example 3. *For the language L given by the DTD*

$$r \to (c + b)^* \,, \quad c \to ab^*a \,,$$

the flattening $\mathrm{flat}(L)$ given by

$$r\big(c(a\overline{a}(b\overline{b})^*a\overline{a})\overline{c} + b\overline{b}\big)^*\overline{r}$$

is recognized by an AC^0 family of streaming circuits. [2]

[1]The proofs of claims marked with (♠) are in the appendix

[2]This can be verified using, for instance, an on-line tool available at `http://paperman.cadilhac.name/sage/`.

As regular languages in AC^0 have an exact logical characterization, and membership in AC^0 is decidable, one could decide whether the flattening of a given regular bounded-depth tree language can be recognized by an AC^0 family of circuits. To explain it in more detail, we need to recall some classical results in descriptive complexity. Then, we shall look at practical fragments of the DTD formalism that guarantee the existence of AC^0 streaming circuit families.

First order logic and constant-depth circuits. We consider first order logic over words, encoded as relational structures over universe $\{0, \dots, n-1\}$ where n is the length of the word. Formulas are generated by the first order logic grammar with two kinds of atomic predicates: the *letter predicates* of the form $\mathbf{a}(x)$ that are true if and only if the position x in the word is labelled by a, and *numerical predicates* which are predicates speaking about the word stripped of labels. For conciseness, we also allow numerical constants min and max for the first and last positions in the word.

Example 4. *The language*

$$\{a^n b^n \mid n \in \mathbb{N}\}$$

is defined by the formula

$$\max \equiv 1 \bmod 2 \wedge \forall y \; y < \frac{\max}{2} \leftrightarrow \mathbf{a}(y)\,.$$

It is well known that word languages definable in FO with arbitrary numerical predicates are exactly languages in AC^0 (see [14] for instance). The simplest way to translate an FO sentence φ into a constant-depth circuit is to introduce a gate for each subformula $\psi(x_1, \dots, x_k)$ and each choice of positions $i_1, \dots, i_k$ in the word. The most external logical symbol in ψ determines the type of the gate: $\vee$, $\wedge$, $\neg$ correspond to OR, AND or NOT gates, and quantifiers are interpreted as disjunctions and conjunctions over all positions of the word. The gate is connected to the gates corresponding to appropriate subformulas of $\psi(x_1, \dots, x_k)$ with variables valuated accordingly. The gates for the letter predicates are simply the binary input gates encoding the input symbols. If P is a numeric predicate, the gate for $P(i_1, \dots, i_k)$ is either constantly 0 or constantly 1, depending on P and $i_1, \dots, i_k$ (this is where we use non-uniformity). The depth of this circuit is bounded by the depth of the formula, seen as a term. The number of gates is bounded by $\|\varphi\| \cdot n^k$, where $\|\varphi\|$ is the number of different subformulas in φ and k is the maximal number of free variables in a subformula.

This construction can be optimized for FO^k, that is, for formulas using (and reusing) only k variables (see [19, 23]). Such a formula can be written in a normal form in which quantification is always of the form

$$\exists x_1 \; \delta(x_1, \dots, x_k) \wedge \psi_2 \wedge \dots \wedge \psi_k$$

such that $\delta(x_1, \dots, x_k)$ is a quantifier-free formula using only numerical predicates, and the set of free variables of ψ_j does not contain x_j. Then, it essentially suffices to have gates for subformulas with at most $k-1$ free variables: for each valuation of variables $x_2, \dots, x_k$, we have an OR gate connected to the ANDs of the gates for ψ_j with the variable x_1 valuated in all ways that make $\delta(x_1, \dots, x_k)$ hold. The size of the resulting circuit is bounded by $\|\varphi\| \cdot n^{k-1}$.

Regular languages and logic. The connection between logic and regular languages is a field of research on its own that takes its root into the celebrated results of McNaughton and Papert [22] and Schützenberger [26], who characterized regular languages of FO[<], that is languages definable in first order logic with the linear order over positions. By extending this result to a slightly more complicated fragment, and by using the parity lower bounds for AC^0, Barrington et al. [1] proved that regular languages in AC^0 are exactly those definable in FO[<, MOD]; that is, in first order logic with (strict) order and the unary modulo predicates of the form $x \equiv r \bmod q$ for arbitrary $r, q \in \mathbb{N}$. Furthermore, this class of regular languages has decidable membership thanks to its algebraic characterisation.

Thus, we get the following corollary from Theorem 1.

Corollary 1. *For a given regular tree language one can decide if its flattening can be recognized with an* AC^0 *streaming circuit family; a recognizing streaming circuit for a given block size can be constructed effectively.*

Checking whether a regular word language is definable in FO[<, MOD] is PSPACE-complete [8]. The algorithm to construct a circuit from an automaton runs in time linear in the size of the *syntactic monoid* of the recognized language, and is therefore efficient as long as the syntactic monoid is not too large. In general, the syntactic monoid has size at most exponential in the size of the minimal automaton recognizing the language.

A more practical approach to providing AC^0 streaming circuits is to define a subclass of DTDs that are directly transformable into AC^0 streaming circuit families. In order to identify such a subclass, we first show that for bounded-depth tree languages, definability in FO is equivalent to FO[<]-definability of the flattening.

FO-definable tree languages. First order logic over trees uses the letter predicates $\mathbf{a}(x)$ and navigational predicates child(x, y), descendant(x, y), nextSibling(x, y) and followingSibling(x, y), which hold if and only if y is respectively a child, descendant, the next sibling or a following sibling of x.

Example 5. *The language of trees over unary alphabet with only one branch is defined by the formulas*

$$\forall x, y \; \textit{descendant}(x, y) \vee \textit{descendant}(y, x)\,.$$

Note that the flattening of this language is exactly the one given in the previous example (with $b = \overline{a}$*).*

We begin with a lemma which shows that, assuming bounded depth, the tree structure can be recovered from the flattening with FO[<] formulas. In the flattening, we think of the positions with the opening tags as the ones representing the nodes of the tree.

LEMMA 1. *For all $d > 0$, there exist* FO[<] *formulas*

$$tree_d(x,y)\ ,\ forest_d(x,y)$$

expressing, respectively, that the segment from x to y is the flattening of a tree of depth at most d, and the segment between x an y is a concatenation of the flattenings of trees of depth at most d, as well as FO[<] *formulas*

$$child_d(x,y),\ desc_d(x,y),\ next_d(x,y),\ following_d(x,y)$$

expressing (over flattenings of trees of depth at most d) that the node represented by position x and the node represented by position y are in relation child, descendant, next sibling, and following sibling, respectively.

PROOF. The formulas tree_d and forest_d are defined by mutual recursion. Let $\text{tree}_0(x,y) = \text{forest}_d = \text{false}$. For $d > 0$, the formula $\text{forest}_d(x,y)$ expresses that each position between x and y a has a matching position between x and y such that the corresponding segment is the flattening of a tree of depth at most d,

$$\begin{aligned}\text{forest}_d(x,y) :=\ & x < y \wedge \\ & \wedge\ \forall u \in (x,y)\ \exists v \in (x,y)\ \big(\text{tree}_d(u,v) \vee \text{tree}_d(v,u)\big)\,,\end{aligned}$$

and $\text{tree}_d(x,y)$ checks that x and y are labelled by matching tags and the segment between them is a concatenation of the flattenings of trees of depth at most $d-1$,

$$\text{tree}_d(x,y) := \Big(\bigvee_{a\in\Sigma} \mathbf{a}(x) \wedge \overline{\mathbf{a}}(y)\Big) \wedge \text{forest}_{d-1}(x,y)\,.$$

It is not difficult to verify that tree_d and forest_d indeed define, respectively, flattenings of trees of depth at most d and concatenations of such. We use tree_d to define the descendant relation:

$$\text{desc}_d(x,y) := \exists x'\ \text{tree}_d(x,x') \wedge y \in (x,x') \wedge \Big(\bigvee_{a\in\Sigma} \mathbf{a}(y)\Big)\,.$$

The child relation is then easy to define:

$$\begin{aligned}\text{child}_d(x,y) :=\ & \text{desc}_d(x,y)\ \wedge \\ & \wedge\ \neg\,\exists z \in (x,y)\ \text{desc}_d(x,z) \wedge \text{desc}_d(z,y)\,.\end{aligned}$$

Expressing the following sibling relation is straightforward once we have the child formula:

$$\begin{aligned}\text{following}_d(x,y) :=\ & x < y\ \wedge \\ & \wedge\ \exists z\ \text{child}_d(z,x) \wedge \text{child}_d(z,y)\,.\end{aligned}$$

Finally, the formula

$$\begin{aligned}\text{next}_d(x,y) :=\ & \text{following}_d(x,y)\ \wedge \\ & \wedge\ \neg\,\exists z \in (x,y)\ \text{following}_d(x,z)\end{aligned}$$

defines the next sibling relation in the usual way. □

From Lemma 1 and Theorem 1, we get the following result.

THEOREM 3. *Let L be a bounded-bounded depth tree language. Then, L is* FO*-definable if and only if* flat(L) *is* FO[<]*-definable. In consequence, flattenings of* FO*-definable tree languages are recognized by* AC^0 *streaming circuit families; the recognizing streaming circuit for a given block size can be constructed effectively.*

PROOF. The formula defining the flattening of L is obtained by taking the conjunction of $\text{tree}_d(\min,\max)$ and the formula defining L with each occurrence of child, descendant, next-sibling, and following-sibling replaced with the appropriate formula given by Lemma 1.

For the converse implication, note that we can rewrite each FO[<]-formula over the flattenings so that quantification is in one of the following forms:

$$\exists x^o\ \Big(\bigvee_{a\in\Sigma} \mathbf{a}(x^o)\Big) \wedge \varphi\,, \qquad \exists x^c\ \Big(\bigvee_{a\in\Sigma} \overline{\mathbf{a}}(x^c)\Big) \wedge \varphi\,;$$

the resulting formula is at most exponentially larger. Then, each variable has its type, opening or closing. We now rewrite each atomic predicate for all possible types of variables. For an opening variable x^o, $\mathbf{a}(x^o)$ remains unchanged and $\overline{\mathbf{a}}(x^o)$ is rewritten as *false*; for a closing variable x^c, $\mathbf{a}(x^c)$ is rewritten as *false*, and $\overline{\mathbf{a}}(x^c)$ as $\mathbf{a}(x^c)$. For a closing variable x^c and an opening variable y^o, the atomic formula $x^c < y^o$ is rewritten as $\text{right}(x^c,y^o)$, where $\text{right}(x,y)$ is the formula

$$\begin{aligned}\exists z \exists z'\ & (\text{descendant}(z,x) \vee z = x)\ \wedge \\ & \wedge\ \text{followingSibling}(z,z')\ \wedge \\ & \wedge\ (\text{descendant}(z',y) \vee z' = y)\,.\end{aligned}$$

Similarly, the formulas

$$\begin{aligned}& \text{descendant}(x^o,y^o) \vee \text{right}(x^o,y^o)\,, \\ & \text{descendant}(x^o,y^c) \vee \text{right}(x^o,y^c)\ \vee \\ & \qquad\qquad \vee\ \text{descendant}(y^c,x^o) \vee x^o = y^c\,, \\ & \text{right}(x^c,y^c) \vee \text{descendant}(y^c,x^c)\end{aligned}$$

are used for the remaining three cases. We obtain a formula of FO on trees, easily seen to be equivalent to the original formula of FO[<] on flattenings. □

Theorem 3 gives an effective sufficient condition for the existence of an AC^0 streaming circuit family for the flattening: as FO[<] definability is decidable for regular languages, so is FO-definability for bounded-depth regular tree languages. We remark that for regular tree languages of unbounded depth decidability of FO-definability is a major open problem [5].

The condition given by Theorem 3 is not necessary. As shown in the next example, capturing the entire class of bounded-depth regular tree languages admitting AC^0 streaming circuit families for the flattenings would require intricate artificial syntactic restrictions over the basic formalism, with an unclear gain in expressivity.

EXAMPLE 6. *Consider the following two DTDs:*

$$r \to (aa)^*\ ,\ a \to (bb)^*\ ; \qquad r \to (aa)^*\ ,\ a \to (bbb)^*\,.$$

The flattening of the language given by the left one is AC^0*, whereas for the right one it is not.*

The argument in Theorem 3 does not give good complexity bounds: the FO formula for the flattening has size linear in the original formula (and exponential in the depth), but the automaton constructed from the formula, needed to invoke Theorem 1, may have non-elementary size. And even if there was a more efficient way to do it, FO is not a natural schema definition language. A desirable language should be a natural fragment of a known schema definition language. We discuss such a fragment in the following subsection.

We finish this subsection with a remark that without the bounded-depth assumption one can recognize flattenings of FO-definable tree languages in TC^0 in the random-access model. Recall that TC^0 is defined like AC^0, but except that Majority gates can also be used.

Proposition 4. *Let L be an* FO*-definable language of trees. Then* $\mathrm{flat}(L)$ *is in* TC^0. (♠)

Even the flattening of the set of all trees is TC^0-hard, but not all flattenings of regular languages of unbounded depth are: for instance the language of trees with only one branch over unary alphabet is FO-definable and its flattening is in AC^0 (see Example 4).

A practical formalism for validation in constant-depth. A natural formalism allowing validation with constant-depth streaming circuits can be obtained by restricting the productions in DTDs to FO[<]-definable languages. Over words, being FO[<]-definable is equivalent to being definable with a *star-free* regular expression [22]; that is, an expression build from symbols from the alphabet and the empty set by means of concatenation and all Boolean operations, including complement.

Corollary 2. *Each language $L \subseteq \mathrm{Trees}(\Sigma)$ defined by a non-recursive DTD with star-free productions can be defined in* FO *on trees, and so can be recognized by an* AC^0 *streaming circuit family.*

Proof. For each $a \in \Sigma$ there is an FO[<] sentence φ_a defining the word language generated by the production for a. As the DTD is non-recursive, all generated trees have depth at most $|\Sigma|$. We define L with the formula

$$\exists x\, \mathbf{r}(x) \wedge \forall y\, \neg \mathrm{descendant}(y,x) \wedge \forall z \bigwedge_{a\in\Sigma} \mathbf{a}(z) \rightarrow \widehat{\varphi}_a(z)\,,$$

where r is the root label of the DTD, and the formula $\widehat{\varphi}_a(z)$ is obtained from the sentence φ_a by replacing each occurrence of the predicate $<$ with the predicate followingSibling and restricting all quantifiers to the children of z; that is, $\exists y\, \psi$ is replaced with $\exists y\, \mathrm{child}(z,y) \wedge \psi$ and $\forall y\, \psi$ is replaced with $\forall y\, \mathrm{child}(z,y) \rightarrow \psi$ (assuming that variable z is not used in φ_a). □

The popular class of nested-relational DTDs is a special case, which admits constant-depth streaming circuits with particularly good parameters.

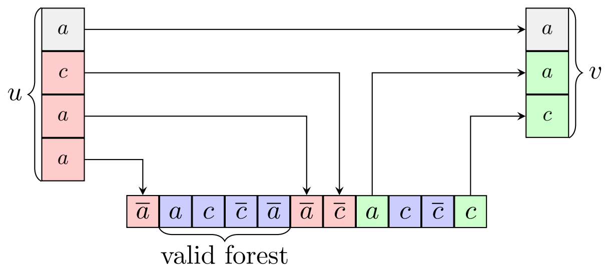
Theorem 4. *Each language $L \subseteq \mathrm{Trees}(\Sigma)$ defined by a nested-relational DTD can be recognized by an AC^0 streaming circuit family with feedback $\mathcal{O}(d \cdot |\Sigma|)$, depth $\mathcal{O}(d)$ and size $\mathcal{O}(d^3 \cdot |\Sigma|^2 \cdot n^2)$ for block size n, where $d \le |\Sigma|$ is the maximal depth of trees in L.*

Proof. We shall directly construct a streaming circuit, using FO formulas over separate blocks of the input word as an intermediate formalism. Since the language is defined by a nested-relational DTD, its depth is bounded by some $d \le |\Sigma|$. Before we look at the DTD any further, we construct a circuit that for each position x in the block computes $\mathit{open}(x) \in \Sigma^{\le d}$, the sequence of unmatched opening tags in the prefix of the entire input word up to (and including) position x. Note that $\mathit{open}(x)$ is equal to the sequence of labels on the path from the root to the node corresponding to x in the encoded tree, including this node if the tag of x is opening, and not including it otherwise. As the feedback we shall use $\mathit{open}(\min -1)$, where $\min -1$ is the last position of the previous block. We use the padding symbol \$ (encoded as a sequence of zeros) to fill up $\mathit{open}(\min -1)$ to d symbols.

First, we compute the values of the formula $\mathrm{forest}_d(x,y)$ from Lemma 1 for all positions within the block. Note that $\mathrm{forest}_d(x,y)$ has quantifier rank $\mathcal{O}(d)$ and uses only 3 variables. Its size is exponential if the recursive definition is unravelled, but it has only $\mathcal{O}(d+|\Sigma|)$ different subformulas. Thus, the standard translation for formulas with 3 variables gives a streaming circuit of depth $\mathcal{O}(d)$ and size $\mathcal{O}\big((d+|\Sigma|)\cdot n^2\big)$; the circuit has n^2 output gates, one for each pair of values of x and y. From now on, we shall treat $\mathrm{forest}_d(x,y)$ as an atomic formula.

Similarly, we can assume the existence of atomic formulas $\overline{f}_p(x)$ for $p = 1,2,\dots,d$ expressing that position x has the closing tag corresponding to the p-th letter stored in the feedback. Their values for all x can be computed with a circuit of depth $\mathcal{O}(1)$ and size $\mathcal{O}(d \cdot |\Sigma| \cdot n)$.

How does $v = \mathit{open}(x)$ depend on $u = \mathit{open}(\min -1)$? The situation can be illustrated as follows:


u
a
c
a
a
v
a
a
c
$\bar a$ a c $\bar c$ $\bar a$ $\bar a$ $\bar c$ a c $\bar c$ c
valid forest

We express this with formulas

$$\varphi_{i,a}(x)$$

for $i = 1,2,\dots,d$, and $a \in \Sigma$, saying that the i-th symbol of $\mathit{open}(x)$ is a, assuming that the feedback stores $\mathit{open}(\min -1)$. The formula is a disjunction over $0 \le j \le k, \ell \le d$ with $\ell \ge i$ of conditions saying that

- the first \$ in the feedback is at position $k+1$,
- there exist positions $x \ge y_\ell > y_{\ell-1} > \cdots > y_{j+1}$ with opening tags,
- there exist positions $y_{j+1} > z_{j+1} > z_{j+2} > \cdots > z_k$ with closing tags $\overline{f}_{j+1}, \overline{f}_{j+2}, \dots, \overline{f}_k$, such that
- all segments between these positions (including $\min-1$ and x) are flattenings of forests, and
- $a = f_i$ if $i \le j$, and $a(y_i)$ if $i > j$,

where f_p is the symbol in the p-th feedback gate.

This resulting formula uses $\mathcal{O}(d)$ quantifiers and can be written with 2 variables only. It has has $\mathcal{O}(d^3)$ different subformulas (with $\text{forest}_d(x,y)$ and $\overline{f}_p(x)$ treated as atomic formulas). Thus, the standard translation gives a streaming circuit of depth $\mathcal{O}(d)$ and size $\mathcal{O}(d^3 \cdot n^2)$ for block size n; the circuit has n output gates, one for each value of x. Note that we cannot use the optimized construction giving linear-size circuit, because the formula forest_d does not define a numerical predicate; that is, it depends on the labels of positions.

The $d \cdot |\Sigma|$ circuits for formulas $\varphi_{i,a}(x)$ compute $\mathit{open}(x)$ for all x. If for some x and i we have 0 for all a's, it means that $\mathit{open}(x)$ has less then i symbols. Note that the total size of the constructed circuits is $\mathcal{O}(d^4 \cdot |\Sigma| \cdot n^2)$.

Local correctness of the encoding can be checked by verifying that $\mathit{open}(x)$ is nonempty throughout the computation; as soon as it becomes empty, the remaining positions in the block should store the padding symbol \$, and it should be the last block. All this can be tested with a circuit of depth $\mathcal{O}(1)$ and size $\mathcal{O}(n^2)$.

Once we have built the circuit computing $\mathit{open}(x)$ for each position x, we have access to the label of the parent of each node whose closing tag is within the block: it is the last letter of $\mathit{open}(x)$. The labelling restrictions enforced by the DTD can be expressed with the formula

$$\forall x \in [\min -1, \max) \\ \bigwedge_{a\in\Sigma} \big(\mathbf{a}(x) \to \bigvee_{c \in N(a)} \mathbf{c}(x+1)\big) \wedge \\ \wedge \bigwedge_{a,b\in\Sigma} \big(\mathit{open}(x) \in \Sigma^* a \$^* \wedge \overline{\mathbf{b}}(x) \to \bigvee_{c\in N(a,b)} \mathbf{c}(x+1)\big)\,.$$

where $N(a)$ is the set of labels that can succeed the opening tag a, and $N(a,b)$ is the set of labels that can succeed the closing tag $\overline{b}$ in the scope of tag a. Both sets are determined by the production for a. Assume the production is

$$a \to \widehat{a}_1 \widehat{a}_2 \dots \widehat{a}_k\,.$$

There are two cases, depending on whether there is i such that $\widehat{a}_i$ is either a_i^+ or a_i. If there is such an i, let us take the minimal one. Then, $N(a) = \{a_1, a_2, \dots, a_i\}$. If there is no such i, $N(a) = \{a_1, a_2, \dots, a_k\} \cup \{\overline{a}\}$. The set $N(a, a_j)$ is characterised analogously: if there is $i > j$ such that $\widehat{a}_i$ is either a_i^+ or a_i, for the minimal such i we have $N(a,a_j) = \{a_j, a_{j+1}, \dots, a_i\}$ for $\widehat{a}_j$ equal to a_j^* or a_j^+, and $N(a,a_j) = \{a_{j+1}, \dots, a_i\}$ for $\widehat{a}_j$ equal to $a_j?$ or a_j. Similarly, if no such i exists, $N(a,a_j)$ is either $\{a_j, a_{j+1}, \dots, a_k\} \cup \{\overline{a}\}$ or $\{a_{j+1}, \dots, a_k\} \cup \{\overline{a}\}$, depending on $\widehat{a}_j$.

Note that to evaluate the formula we need access to $\mathit{open}(\min -1)$ and the tag at position $\min -1$. We have already pointed out that $\mathit{open}(\min -1)$ is included in the feedback; now we see that also the label at position $\min -1$ should be a part of the feedback. Assuming unary encoding of letters, the size of the feedback is $\mathcal{O}(|\Sigma|^2)$. The standard translation of the formula gives a circuit of depth $\mathcal{O}(1)$ and size $\mathcal{O}\big((|\Sigma|^2 + d \cdot |\Sigma|) \cdot n\big)$. The output of the circuit is used to propagate the information about the lack of error so far, which is done by means of a designated error feedback gate.

Combining the two stages we obtain a circuit of depth $\mathcal{O}(d) = \mathcal{O}(|\Sigma|)$ and size $\mathcal{O}(d^3 \cdot |\Sigma|^2 \cdot n^2)$. $\square$

Let us remark that the construction can be extended to productions with thresholds $a^{\ell..k}$, $a^{\geq k}$, at the cost of including more information in the feedback: for each label in $\mathit{open}(y)$ we would need the number of its repetitions among its siblings so far (up to threshold k).

5. WIRE-LINEAR CIRCUITS

While having an AC^0 streaming circuit family guarantees depth independent of the block size, it is still possible that the number of gates and the number of wires makes implementation for larger block sizes unreasonable. We now turn to *wire-linear circuit families*, WLAC^0; that is, bounded-depth circuit families in which the number of wires (and thus the number of gates) grows linearly with the size of the input (or block in case of streaming circuits). As for AC^0, WLAC^0 has been studied and regular languages in WLAC^0 are characterized.

Regular languages in WLAC^0. The logical characterization of regular languages in WLAC^0 is given by the following result, extending a similar characterization of regular languages with a *neutral letter* in WLAC^0 [17].

THEOREM 5 ([23]). *A regular language is in* WLAC^0 *if and only if it is definable in* $\mathrm{FO}^2[+1, <, \mathrm{MOD}]$.

Note that the signature includes the successor relation $+1$, which cannot be defined from $<$ with just 2 variables.

Unlike for AC^0, both directions are involved. The lower bound relies on an effective algebraic characterization of languages definable in $\mathrm{FO}^2[+1, <, \mathrm{MOD}]$ from [9] (which also makes definability decidable) and the fact that the language $(c + ac^*b)^*$ is not in WLAC^0 [17]. The upper bound, which we care mostly about, uses a clever circuit construction for the *prefix function* [7]. For completeness, we sketch this construction and explain how to use it to construct a WLAC^0 circuit family from a formula of $\mathrm{FO}^2[+1, <, \mathrm{MOD}]$. To get a WLAC^0 streaming circuit family we use Theorem 1.

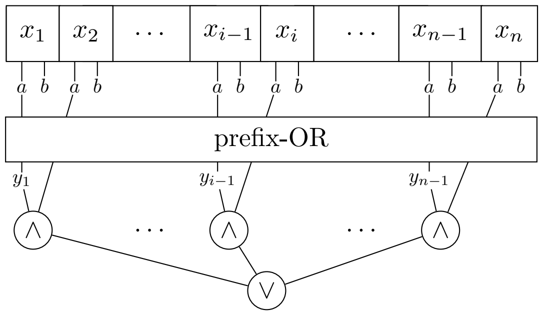
Consider the language $\Sigma^* a \Sigma^* a \Sigma^*$ of words with at least two letters a. It can be defined by the formula

$$\exists x\ \mathbf{a}(x) \wedge \exists y\ y < x \wedge \mathbf{a}(y)$$

The standard translation from FO, which introduces a gate for each subformula with free variables valuated in all possible ways, gives a circuit of quadratic size. The optimized construction for FO^2 formulas gives a circuit which linearly many gates, but quadratically many wires: for each value of x we have an OR gate connected to the circuits for $\mathbf{a}(y)$ for all $y < x$.

To obtain a wire-linear circuit, we use *prefix functions*. The prefix-OR is a function $f\colon \{0,1\}^n \to \{0,1\}^n$ such that $f(u)_i = \bigvee_{j \leq i} u_j$; suffix-OR, prefix-AND, and suffix-AND are defined similarly. A WLAC^0 circuit for prefix-OR is constructed by evaluating prefix-OR naïvely (with quadratically many wires) over the ORs of size-$\sqrt{n}$ blocks, and then over each block separately with the additional knowledge of the bit computed by the first stage for the previous block. If we use a separate circuit for each block, we get a circuit with $\mathcal{O}(n\sqrt{n})$ wires. To avoid this we note that we need to compute the prefix-OR only for the single block where 0's switch to 1's in the prefix-OR for block ORs. The remaining prefix and suffix functions can be computed similarly. For more details we refer to the original article [7].

Coming back to our example, a WLAC^0 circuit for $\Sigma^* a \Sigma^* a \Sigma^*$ can be obtained by computing the prefix-OR of *being letter* a and checking if there exists an input gate which contains a such that the prefix-OR for the previous position evaluates to 1:



This construction can be nested: for the language of words having at least k occurrences of a, one uses k prefix-OR circuits with n inputs interleaved with k layers of AND gates of fan-in 2, with the last layer of AND gates connected to a single OR gate. The size of the resulting circuit is therefore $\mathcal{O}(kn)$ and is independent from the size of the input alphabet.

To build a circuit for an arbitrary FO^2 formula we proceed by structural induction over formulas in the classical normal form. The basic cases are unary predicates $\mathbf{a}(x)$ and $x \equiv r \bmod q$, for which the naïve construction gives wire-linear circuits. As WLAC^0 is closed under Boolean connectives, the only difficulty in the inductive step is the quantification. In the normal form, the quantification is always of the form $\exists y \; \delta(x, y) \wedge \varphi(y)$ where $\delta(x, y)$ only uses predicates $x < y$ and $x = y + k$ for $k \in \mathbb{Z}$. We deal with it like in the example above, by computing prefix functions for $\varphi(1), \ldots, \varphi(n)$ and then for each x adding an OR gate wired appropriately to the outputs of $\varphi(1), \ldots, \varphi(n)$ and the prefix functions; details can be adapted from [17] or found in [23].

Tree languages and* WLAC^0 *validation. Using the described results on regular languages in WLAC^0, and Theorem 1, we obtain the following corollary.

Corollary 3. *Given a regular bounded-depth tree language L, one can decide if* $\mathrm{flat}(L)$ *can be recognized with a* WLAC^0 *streaming circuit family; a recognizing streaming circuit for a given block size can be constructed effectively.* □

XML flattenings of the sets of trees of depth 1 and 2 (over a singleton alphabet), that is, languages $(ab)^*$ and $(a(ab)^*b)^*$, are in WLAC^0, but for depth 3 the flattening is the language $(a(a(ab)^*)^*b)^*$, which is not in WLAC^0. Hence, unlike for full FO, the two-variable fragment of FO over trees is not captured by WLAC^0 (and thus by FO^2 over words). In fact, it is not easy to imagine a nontrivial schema formalism that guarantees recognizability with a WLAC^0 streaming circuit family. We shall eventually propose such a formalism, but for now we shift the perspective and ask: what if we change the way trees are encoded as words?

We propose an encoding giving even more information than the XML encoding: the path-from-the-root encoding. Trees of depth d over alphabet Σ are encoded as words over alphabet

$$\Sigma \cup \overline{\Sigma} \cup \{\$\}.$$

For $0 < i \leq d$, $u \in \Sigma^{i-1}$, a tree t with root $a \in \Sigma$, and children that are trees $t_1, \ldots, t_k$ we set

$$\Delta^d_u(t) = ua\$^{d-i} \cdot \Delta^d_{ua}(t_1) \cdots \Delta^d_{ua}(t_k) \cdot u\overline{a}\$^{d-i}$$

and let the *path-from-the-root* encoding of tree t be

$$\Delta^d(t) = \Delta^d_\varepsilon(t)\,.$$

For this new encoding, we still have that the flattening

$$\Delta^d(L) = \big\{\Delta^d(t) \;\big|\; t \in L\big\}$$

of a regular bounded-depth tree language L is a regular language of words. Moreover, the correctness of the encoding can be checked by a WLAC^0 streaming circuit family. For the remaining, we will simply denote by $\Delta(L)$ the encoding of L, assuming that the parameter d is clear from the context.

Lemma 2. *Let* $\mathrm{Trees}_d(\Sigma)$ *be the set of trees over* Σ *of depth at most* d. *The language* $\Delta\big(\mathrm{Trees}_d(\Sigma)\big)$ *is recognized by a* WLAC^0 *streaming circuit family with feedback* $\mathcal{O}(d \cdot |\Sigma|)$, *depth* $\mathcal{O}(1)$, *and* $\mathcal{O}\big(|\Sigma| \cdot (n + d)\big)$ *wires.*

Proof. To encode consecutive tags in the flattening, the path-from-the-root encoding uses blocks of d consecutive symbols: at most d symbols on the path to the root, padded to length d with symbol \$. These blocks will be called *d-slabs*. For simplicity, in the following we assume that n is divisible by d, so that d-slabs fit exactly into blocks processed by the circuit. If this is not the case, we can adapt the construction by passing the previous incomplete block in the feedback, together with the length $r < d$ of the passed fragment, encoded in unary. This increases the number of feedback gates by $\mathcal{O}(d)$. All the congruences used in the following constructions can be easily adjusted to take in the account the fact that the d-slabs in the block are shifted by r.

The feedback consists of the last d-slab of the previously processed block, and a special error gate that passes the information whether an error has been encountered so far. In the first block, we assume that the initial feedback encodes a d-slab consisting only of symbols \$. As such d-slabs will never appear again throughout the computation, from this feedback we can recognize that the first block is being processed. Henceforth we assume that the circuit has the access to symbols at positions between $\min -d$ and $\min -1$, or to the information that these positions do not exist.

Let us introduce an auxiliary formula $\mathrm{last}(x)$ expressing that x is the last position with a non-padding symbol within its d-slab:

$$\mathrm{last}(x) := \neg\$(x) \wedge \big(\$(x+1) \vee x \equiv d-1 \bmod d\big)\,.$$

We can compute this formula for all positions x using $\mathcal{O}(n)$ gates and depth $\mathcal{O}(1)$.

We proceed to verifying that the block's description is correct. For this, we check that certain conditions hold for every position $x \in [\min -d, \max -d)$ (resp. $x \in [0, \max -d)$ during processing the first block) using a formula, whose encoding as a circuit will be straightforward. We first express that the padding symbols behave as expected:

$$\begin{aligned} x \equiv 0 \bmod d \rightarrow\ & \neg\$(x)\,, \\ \$(x) \wedge \neg\$(x+1) \rightarrow\ & x \equiv d-1 \bmod d\,, \\ \$(x) \wedge \neg\$(x+d) \rightarrow\ & \mathrm{last}(x-1)\,. \end{aligned}$$

The first implication asserts that no d-slab contains the \$ symbol at its front. The second one asserts that after any \$ symbol, we necessarily have \$ symbols up to the end of the d-slab. The third one asserts that the only \$ symbol that can be replaced by a letter in the next d-slab is the first one. Finally, we introduce the following implications for all $a \in \Sigma$:

$$\begin{aligned} \mathbf{a}(x) \rightarrow\ & \overline{\mathbf{a}}(x+d) \vee \big(\mathbf{a}(x+d) \wedge \neg\mathrm{last}(x+d)\big)\,, \\ \overline{\mathbf{a}}(x+d) \rightarrow\ & \mathrm{last}(x+d) \wedge \mathbf{a}(x)\,. \end{aligned}$$

The first implication asserts that an opening tag either changes to the matching closing tag or remains the same and another symbol is added after it. In particular, an opening tag is never replaced by \$. The second implication asserts that a closing tag has to be produced from the last position of the previous d-slab. In particular, in the next d-slab it can be replaced by an opening tag or a \$ symbol, but not by a closing tag. The described formula can be turned directly into a wire-linear streaming circuit with a single gate of unbounded fan-in, connected to $|\Sigma| \cdot n$ subcircuits of size $\mathcal{O}(1)$ and constant fan-in.

The circuit described above verifies that the description of the block is correct. The error gate passed as the feedback to the next iteration is its conjunction with the error gate received in the feedback from the previous iteration. The acceptor circuit just checks that no error has occurred. $\square$

Even if some structural properties of the input trees are accessible under the new encoding, WLAC^0 circuits still fail to capture FO^2 definable tree languages, as shown in the next example.

EXAMPLE 7. *Let L be the language given by the DTD*

$$a \rightarrow b^*\,, \quad b \rightarrow (c+d)^*$$

restricted to trees in which one of the root's children has only c-children. It is clearly definable in FO^2. *The flattening $\Delta(L)$ is (up to relabelling) the language*

$$(a(b(c\overline{c} + d\overline{d})^*\overline{b})^*\overline{a} \,\cap\, \Sigma^* b(c\overline{c})^*\overline{b}\Sigma^*\,,$$

which is not in WLAC^0.

This latter example works mainly because the language is not defined by a DTD. Of course, it is hopeless to believe that we can have a good streaming circuit for any bounded-depth DTD, but what if we restrict the productions to $\mathrm{FO}^2[<]$-definable languages? As it turns out we can still get a tree language whose path-from-the-root encoding is not recognizable with a WLAC^0 streaming circuit family.

EXAMPLE 8. *For the language L given by the DTD*

$$r \rightarrow (a)^*\,, \quad a \rightarrow b^*c^*b^*\,,$$

the flattening $\Delta(L)$ is (up to relabelling) the language

$$r(a(b\overline{b})^*(c\overline{c})^*(b\overline{b})^*\overline{a})^*\overline{r}\,,$$

which is not in WLAC^0.

What we can tackle are nested-relational DTDs.

PROPOSITION 5. *Let L be a tree language defined by a nested-relational DTD. Then $\Delta(L)$ is definable in* $\mathrm{FO}^2[<,+1,\mathrm{MOD}]$ *and is recognized by a* WLAC^0 *streaming circuit family with feedback $\mathcal{O}(|\Sigma|^2)$, depth $\mathcal{O}(1)$, and $\mathcal{O}(|\Sigma|^2 \cdot n)$ wires for block size n.* $\square$

PROOF. It suffices to combine the construction from Lemma 2, guaranteeing correctness of the encoding, with the second stage of the construction in Theorem 4. $\square$

Unlike for Theorem 4, we cannot directly extend the construction to DTDs with more general thresholds.

EXAMPLE 9. *For the language L defined by the DTD*

$$r \rightarrow a^*\,, \quad a \rightarrow bb\,, \quad b \rightarrow c^*$$

the flattening $\Delta(L)$ is (up to relabelling) the language

$$r(ab(c\overline{c})^*\overline{b}b(c\overline{c})^*\overline{b}\overline{a})^*\overline{r}\,,$$

which is not in WLAC^0.

Using a new encoding may seem to be an easy way out, since we add exactly the information needed to validate nested-relational DTDs with circuits having good parameters. However, it is possible to benefit from this solution even without using the path-from-the-root encoding. If we have control over the design of the schema, by adjusting the tags we can assume that each label uniquely determines the label of the father. This does not seem to restrict the practical scope of nested-relational DTDs, and provides a nontrivial schema formalism guaranteeing WLAC^0 streaming circuit families. If we cannot or would not modify the schema nor change the encoding into path-from-the-root, we can enrich the given encoding before feeding it to the validating streaming circuit. The enriching can be done by a fixed transducer (dependent only on the alphabet, not the schema itself), implemented with an AC^0 streaming circuit family with additional outputs. This should be viewed as a (rather complicated) fixed device that could be optimized and implemented in hardware once for all. Meanwhile, the proper validation stage can be realised with a reprogrammable hardware device of adapted size, that can be readjusted as the schema changes.

A *streaming circuit with output* with input alphabet Σ, output alphabet Γ, block size n, and feedback size m is like a streaming circuit, except that it has two kinds of output gates: immediate and pass-on. The output word over a given input word is obtained by concatenating the values on the immediate output gates for subsequent blocks of the input word; the values on the pass-on output gates are sent to the feedback gates when the next block is processed. The immediate output gates encode letters from Γ in unary, just as the input letters are encoded: i-th letter is encoded as $0^{i-1}10^{|\Gamma|-i}$. The output value is returned only if the acceptor circuit returns 1; if it returns 0, the output is undefined.

Proposition 6. *Let Σ be a finite alphabet and let $d \in \mathbb{N}$. For trees over Σ of depth at most d, one can compute the path-from-the-root flattening from the XML flattening with a streaming circuit with output that has feedback $\mathcal{O}(d \cdot |\Sigma|)$, depth $\mathcal{O}(d)$, and size $\mathcal{O}\big(d^4 \cdot |\Sigma| \cdot n^2\big)$ for block size n.*

Proof. The first stage of the construction in the proof of Theorem 4 gives almost the circuit we need: it remains to append $\bar{a}$ to *open*(x) for positions x labelled with $\bar{a}$; this does not influence the bounds. □

6. CONCLUSIONS

We have introduced streaming circuits, which model parallel processing of streamed data in a way compatible with the schema validation task. We have shown that general results on the circuit complexity of regular languages can be used directly to reason about the existence of good streaming circuits for bounded-depth regular tree languages, giving effective but inefficient criteria. For a restricted, but practically crucial, class of languages defined by nested-relational DTDs we have provided a direct construction of circuits with excellent parameters: compositions of a quadratic-size AC^0 circuits dependent only on the depth of trees and the alphabet, and wire-linear AC^0 circuits dependent on the DTD. This construction can be extended easily to schemas with more general thresholds. Extending it further would be very relevant practically.

We have seen how to get a constant-depth polynomial-size streaming circuit from a bounded-depth tree language definable in first order logic. This relies on the fact that FO-definability of word languages is decidable and effective. There is a certain trade-off between the depth of the circuits and the degree of the polynomial bounding their size. We have seen that all FO-definable languages have quadratic circuits, but achieving this requires increasing the depth. It is even possible to achieve near-linear upper bound for these languages by increasing the depth sufficiently (see [18]). However, one may want to optimise the depth at the cost of increasing the degree of the polynomial. This question is related to the famous dot-depth conjecture, which is equivalent to deciding levels of the alternation hierarchy of first-order logic. Indeed, if a language belongs to the k-th level of this hierarchy then it is a finite Boolean combination of regular languages recognized by depth-k circuits. Straubing conjectures that the languages in k-th level of the hierarchy (enriched with the modulo predicates) are exactly the Boolean combinations of regular languages recognized by depth-k circuits [29].

Another related problem is the question of the circuit complexity of word encodings of regular tree languages (of unbounded depth). While there are TC^0-complete and NC^1-complete examples, no good characterizations are known. We conjecture that each regular tree language is either NC^1-complete or in TC^0. It would be very interesting to have an effective characterisation of NC^1-complete regular tree languages such that languages that do not satisfy it are in TC^0. Such a characterization exists in the classical setting of word languages and relies on an algebraic decomposition of finite automata [29]. Since such a decomposition for tree automata is unknown and related to the open question of deciding FO-definability of regular tree languages [5], we believe this question might be very hard.

Checking correctness of the encoding has a huge impact on validation. A way to isolate it is to consider *weak validation*, where the input is assumed to be a correct encoding of a tree. While no tree language of unbounded depth can be validated in constant memory, some can be weakly validated [27, 28]. For instance, for the set of trees whose each a node's leftmost child has label b, we only need to check that each opening a tag is followed by an opening b tag. However, it remains an open question to decide whether a given language can be weakly validated in constant memory. When restricted to bounded-depth tree languages, this question can be seen as a special case of the *separation* question for regular word languages, which has rich bibliography of its own. For instance, separation of regular word languages by $\mathrm{FO}[<]$-definable languages is decidable [13, 24], and similarly for $\mathrm{FO}^2[+1,<]$ [25]. One can use these results as a black box to find good streaming circuits for the weak validation. Unfortunately, the separation abstraction is too powerful for tree languages of unbounded depth: a yet unpublished result by Kopczyński shows that separation of regular tree languages with regular word languages is undecidable under both XML and term encoding [16]. In contrast, the weak validation problem under term encoding is decidable [6], and still open under XML encoding.

7. REFERENCES

[1] David A. Mix Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *J. Computer and System Sciences*, 44(3):478–499, 1992.

[2] G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: a practical study. In *WEBDB 2004*, pages 79–84, 2004.

[3] David Black-Schaffer. *Block-Parallel Programming for Real-Time Applications on Multi-Core Processors*. PhD thesis, Humboldt-Universität zu Berlin, 2008. Available at `http://cva.stanford.edu/people/davidbbs/black-schaffer_thesis_final.pdf`.

[4] David Black-Schaffer and William J. Dally. Block-parallel programming for real-time embedded applications. In *ICPP 2010*, pages 297–306. IEEE Computer Society, 2010.

[5] Mikołaj Bojańczyk, Howard Straubing, and Igor Walukiewicz. Wreath products of forest algebras, with applications to tree logics. *Logical Methods in Computer Science*, 8(3), 2012.

[6] Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer Berlin Heidelberg, 2006.

[7] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Lower bounds for constant depth circuits for prefix problems. In Josep Díaz, editor, *ICALP 1983*, volume 154 of *Lecture Notes in Computer Science*, pages 109–117. Springer, 1983.

[8] Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is pspace-complete. *Theoretical Computer Science*, 88(1):99 – 116, 1991.

[9] Luc Dartois and Charles Paperman. Alternation hierarchies of first order logic with regular predicates. In Adrian Kosowski and Igor Walukiewicz, editors, *FCT 2015*, volume 9210 of *Lecture Notes in Computer Science*, pages 160–172. Springer, 2015.

[10] Tathagata Das, Yuan Zhong, Ion Stoica, and Scott Shenker. Adaptive stream processing using dynamic batch sizing. In Ed Lazowska, Doug Terry, Remzi H. Arpaci-Dusseau, and Johannes Gehrke, editors, *SoCC 2014*, pages 16:1–16:13. ACM, 2014.

[11] Patrick Dymond. Input-driven Languages Are in Log N Depth. *Inf. Process. Lett.*, 26(5):247–250, January 1988.

[12] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Theory of Computing Systems*, 17:13–27, 1984.

[13] Karsten Henckell. Pointlike sets: the finest aperiodic cover of a finite semigroup. *Journal of Pure and Applied Algebra*, 55(1):85 – 126, 1988.

[14] Neil Immerman. Languages that capture complexity classes. *SIAM J. Computing*, 16(4):760–778, 1987.

[15] Rashid Khogali, Olivia Das, and Kaamran Raahemifar. Mobile parallel computing algorithms for single-buffered, speed-scalable processors. In *TrustCom 2013 / ISPA-13 / IUCC-2013*, pages 1832–1839. IEEE Computer Society, 2013.

[16] Eryk Kopczyński. Invisible pushdown languages. *CoRR*, abs/1511.00289, 2015.

[17] Michal Koucký, Pavel Pudlák, and Denis Thérien. Bounded-depth circuits: separating wires from gates. In Harold N. Gabow and Ronald Fagin, editors, *STOC 2005*, pages 257–265. ACM, 2005.

[18] Michal Koucký. Circuit complexity of regular languages. *Theory of Computing Systems*, 45(4):865–879, 2009.

[19] Michal Koucký, Clemens Lautemann, Sebastian Poloczek, and Denis Thérien. Circuit Lower Bounds via Ehrenfeucht-Fraissé Games. In *CCC 2006*, pages 190–201, 2006.

[20] Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Visibly pushdown automata for streaming XML. In *WWW 2007*, pages 1053–1062. ACM, 2007.

[21] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *SIGMOD Record*, 40(4):11–20, 2011.

[22] Robert McNaughton and Seymour Papert. *Counter-Free Automata*. The MIT Press, Cambridge, Mass., 1971.

[23] Charles Paperman. *Circuits booléens, prédicats modulaires et langages réguliers*. PhD dissertation, Université Paris Diderot, 2014. In French.

[24] Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. In Thomas A. Henzinger and Dale Miller, editors, *CSL-LICS 2014*, pages 75:1–75:10. ACM, 2014.

[25] Thomas Place and Marc Zeitoun. Separation and the Successor Relation. In Ernst W. Mayr and Nicolas Ollinger, editors, *STACS 2015*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 662–675, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[26] Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[27] Luc Segoufin and Cristina Sirangelo. Constant-memory validation of streaming XML documents against dtds. In Thomas Schwentick and Dan Suciu, editors, *ICDT 2007*, volume 4353 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2007.

[28] Luc Segoufin and Victor Vianu. Validating streaming XML documents. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *PODS 2002*, pages 53–64. ACM, 2002.

[29] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994.

APPENDIX

Below we include the proofs omitted in the main part of the paper. For convenience we repeat the statements of the propositions.

Proposition 2. *Let $\mathcal{A}$ be a tree automaton with k states recognizing a bounded-depth language $L \subsetneq \mathrm{Trees}(\Sigma)$. One can construct a deterministic automaton with*

$$\mathcal{O}(|\Sigma|^k \cdot 2^{k^2})$$

states recognizing $\mathrm{flat}(L)$.

Proof. As $L(\mathcal{A})$ has bounded depth, each accepting run of $\mathcal{A}$ uses each state at most once on each branch of the input tree. Indeed, if this was not the case, one could construct an arbitrarily deep tree accepted by $\mathcal{A}$ by repeating the part of the tree corresponding to the segment of the branch between two occurrences of the same state. Consequently, $L(\mathcal{A})$ has depth at most k.

Let $\mathcal{B} = (\Sigma, Q, q_0, \delta, F)$ be a deterministic automaton recognizing L obtained from $\mathcal{A}$ by the standard powerset construction; we have $|Q| = 2^k$. The automaton for $\mathrm{flat}(L)$ simulates stack of depth at most k in its states. Its state-space is

$$(\Sigma \times Q)^{\leq k} \cup \{\bot, \top\},$$

where the empty sequence ε is the initial state and $\top$ is the only final state. The transitions are given as follows: upon reading symbol $\sigma \in \Sigma$ in state $\alpha \notin \{\bot, \top\}$,

- if $|\alpha| < k$, move to $\alpha\,(\sigma, q_0)$,
- if $|\alpha| = k$, move to $\bot$;

upon reading symbol $\overline{\sigma} \in \overline{\Sigma}$ in state $\alpha \notin \{\bot, \top\}$,

- if $\alpha = \varepsilon$, move to $\bot$,
- if $\alpha = (\sigma, q_1)$ and $\delta(q_0, q_1, \sigma) \in F$, move to $\top$,
- if $\alpha = (\sigma, q_1)$ and $\delta(q_0, q_1, \sigma) \notin F$, move to $\bot$,
- if $\alpha = \beta\,(\sigma', q')(\sigma, q)$, move to $\beta\,(\sigma', \delta(q', q, \sigma))$,
- if $\alpha = \beta\,(\tau, q)$ with $\tau \neq \sigma$, move to $\bot$;

upon reading any symbol in state $\bot$ or $\top$, move to $\bot$. □

Proposition 4. *Let L be an* FO*-definable language of trees. Then* $\mathrm{flat}(L)$ *is in* TC^0.

Proof Sketch. We essentially repeat the proof of Lemma 1 and Theorem 3, but this time using circuits rather than formulas. In the course of the structural induction, we need to deal with formulas with free variables. We work with words over the alphabet $\{0,1\}^n \times (\Sigma \cup \overline{\Sigma})$ (for sufficiently large n). Over such words we evaluate a formula $\varphi(x_1, x_2, \ldots, x_n)$ by assuming that x_i is assigned the *unique* position that has 1 in the i-th coordinate of its label.

It is sufficient to prove that the formula $\mathrm{tree}(x_i, x_j)$, saying that the infix between position x_i and position x_j is the flattening of a tree, is definable in TC^0: to conclude one simply notes that TC^0 is closed under FO quantification (and Boolean connectives), so all predicates from Lemma 1, and all FO formulas using them, can be defined in TC^0 as well.

For every position $y \in [x_i, x_j]$ which is labelled by an opening tag, we find a position $z \in [y+1, x_j]$ such that z is labelled by the matching closing tag and the number of open tags between y and z is exactly the number of closing tags. This last operation can be done since the Equality can be implemented as the AND of two Majority gates: one that count the number of opening tags compared to the closing one, and conversely the other Majority gate count the closing tags compare to the opening one. □