The Role of Logic and Automata in Understanding Transformers

Anthony W. Lin[1,2][0000−0003−4715−5096] and Pablo Barcelo[3][0000−0003−2293−2653]

[1] Max-Planck Institute for Software Systems, Kaiserslautern, Germany
[2] University of Kaiserslautern-Landau, Kaiserslautern, Germany
[3] Institute for Mathematical and Computational Engineering, Pontificia Universidad Católica de Chile & IMFD Chile & CENIA Chile

Abstract. The advent of transformers has in recent years led to powerful and revolutionary Large Language Models (LLMs). Despite this, our understanding on the capability of transformers is still meager. In this invited contribution, we recount the rapid progress in the last few years to the question of what transformers can do. In particular, we will see the integral role of logic and automata (also with some help from circuit complexity) in answering this question. We also mention several open problems at the intersection of logic, automata, verification and transformers.

Keywords: Transformers · Hard Attention · LTL · Regular Languages.

1 Introduction

Recent years witnessed the unprecedented emergence of Large Language Models (LLMs), which have revolutionized many aspects of our lives. LLMs are based on a new neural network model called *transformers*, which extends the classical feed-forward neural network model via *attention mechanisms* for handling texts of arbitrary lengths. Unlike Recurrent Neural Networks (RNN) — which predated transformers by decades — transformers have proven to be efficiently parallelizable and able to capture long-range dependencies better in practice. Despite the rapid adoption of transformers as a mainstream ML model, some limitations of the transformer model have only been understood in recent years. One good example of such a limitation is to perform *counting* in a text, e.g., determine whether there is an even or an odd number of occurrences of a given token in a text.

In recent years, subareas of theoretical computer science — including logic, automata, and circuit complexity — have featured in the rapid development of the theory of expressivity of transformers (cf. [15]). Such a connection has naturally materialized because transformers are computational models that process texts (i.e., strings) and can be studied just like formal models such as finite-state automata, Turing machines, or logics like first-order and second-order logics on strings. Multiple formal models have been developed by varying the following aspects of transformers: attention mechanisms, positional encodings, precision,

and the so-called "chain of thoughts". Guided by both theory building and experimentation, a picture on the expressive power of transformers has slowly emerged. Although this picture is to date incomplete, a respectable body of works have been produced in the so-called FLaNN (Formal Languages and Neural Networks) community, consisting of logicians, automata theorists, and computational linguists.

Why this article? This article has been written to recount *some* gems that have been discovered at the intersection of logic, automata, circuit complexity, and transformers. That is, we do not aim to be exhaustive. The choices of materials are additionally based on our subjective taste[4]. The intended audience of the article includes researchers in logic, automata, verification and programming languages. In particular, we will mention several open problems, which we believe are worth undertaking in the next years.

Highlight of key results. In its simplest form, a transformer can be understood as a formal model that takes an input *text* (i.e. string) and outputs a *token* (i.e. letter). More formally, a transformer gives rise to a function $f : \Sigma^* \to \Sigma$, for some finite alphabet Σ of tokens. Moreover, one could think of f as a family of formal languages $\{L_a\}_{a\in\Sigma}$, where $L_a := \{w \in \Sigma^* : f(w) = a\}$. This connection underlines the bridge between formal languages and transformers: one can simply study such formal languages L_a generated (or recognized) by transformers.

The first set of results in the paper concerns the expressivity of transformers with *unique hard attention* mechanisms (a.k.a. Unique Hard Attention Transformers, or simply UHAT). Such an attention mechanism — which finds the leftmost value that maximizes the attention score — is a simplification of *softmax attention*, which is used in practice but has proven to be tricky to analyze in theory owing to the use of such real-valued functions as e^x. The first key result that we discuss in the paper is from [2, 17]. It connects formal languages definable in various fragments of first-order logic over strings extended with all numerical predicates (equivalently, subclasses of the circuit complexity class AC^0) and UHAT. In particular, the language

$$\mathrm{PARITY} := \{w \in \{a, b\} : |w|_a \equiv 0 \pmod 2\}$$

is well-known [1] not to be in AC^0, therefore cannot be expressed by UHAT. We cover this in Section 3.

The second set of results concerns the expressivity of transformers with *averaging hard attention* mechanisms (a.k.a. Average Hard Attention Transformers, or simply AHAT). Such an attention mechanism — which averages all values that maximize the attention score (unlike simply taking the leftmost value) — provides another approximation of practical transformers, which use softmax attention. In particular, AHAT is tightly connected to Linear Temporal Logic

[4] Before working on FLaNN, the authors primarily researched in logic, automata theory, automated reasoning, finite model theory, and databases.

extended with counting and the circuit complexity class TC^0. We cover this in Section 4

Finally, we discuss the limitations of both UHAT and AHAT as approximations of practical transformers. In particular, we consider a recent promising direction that restricts AHAT to uniform attention layers (i.e., each position receives the same amount of attention). The resulting model, called AHAT[U], appears to be a good approximation of softmax transformers. We also discuss the distinction between expressibility and trainability in Section 5.

Precision. Real-world transformers are implemented on a specific hardware that allows fixed (bit-)precision and fixed memory. Of course, one can allow more precision and more memory by upgrading the hardware. Therefore, researchers in the theory of transformers has adopted a more practical approach by specifying different precision model on a transformer $\mathcal{T}$:

1. *Fixed* precision: there is a constant c on the allowed number of bits for any computation performed by $\mathcal{T}$.
2. *Logarithmic* precision: the number of allowed bits in the computation of $\mathcal{T}$ on a string of length n is $O(\log n)$.
3. *Polynomial* precision: the number of allowed bits in the computation of $\mathcal{T}$ on a string of length n is $O(n^c)$ for some constant c.
4. *Rational* (resp. *real*) precision: this means rational (resp. real) computation is allowed with an unbounded precision.

Although the distinction is important, it overcomplicates an introductory article. For these reasons, we will assume the last precision model, and note that all of the mentioned results work also for polynomial precision (and often also logarithmic precision).

Notation and assumed background. We assume familiarity with standard results in logic and automata, and their connections to circuit complexity. All required background could be found in the excellent book [11] by Libkin. In particular, we will consider *star-free* languages (i.e. regular languages generated by regular expressions that use concatenation, union, complementation, but no Kleene star), and their equivalent formulation using first-order logic over strings (i.e. over the embedding of strings as logical structures, e.g., *aba* is encoded as the structure with universe $\{1, 2, 3\}$, the order relation $\preceq \subseteq \{1, 2, 3\}^2$, and unary relations $U_a = \{1, 3\}$ and $U_b = \{2\}$ indicating which positions labeled by a and b, respectively). By Kamp's theorem [8], the logic is equivalent to Linear Temporal Logic (LTL). First-order logic characterization of star-free languages can be extended with all numerical predicates to give us a characterization of the circuit complexity class (nonuniform) AC^0, which can be defined by a class of problems that can be solved by a family $\{C_n\}_{n\geq 0}$ of constant-depth polynomial-sized (i.e. polynomial in n) boolean circuits (with unbounded fan-ins), wherein C_n is employed to decide input strings of length n. Note that a k-ary numerical predicate simply means a relation $R \subseteq \mathbb{N}^k$. In the sequel, we also use the

fragment FO[Mon], which restricts the above use of numerical predicates only to *monadic* (i.e. unary) numerical predicates. This is a strict subset of AC^0.

The circuit complexity TC^0 extends AC^0 with majority gates, which effectively allows one to encode all standard arithmetic operations on numbers including addition, multiplication, etc. TC^0 problems are often construed in the FLaNN (Formal Languages and Neural Networks) community as *efficiently parallelizable* problems. Note that TC^0 is a subset of the circuit complexity class NC^1, which contains all problems solvable by families of polynomial-sized circuits of logarithmic depth. It is known that NC^1 contains all regular languages. [It is not known if all regular languages are contained in AC^0]. In turn, NC^1 is a subset of L, i.e., the class of problems solvable in logarithmic space.

2 Formal Models of Transformers

We define several formal models of transformers, which are based on the type of adopted attention mechanisms (i.e. hard or soft attention). We first define these semantically, and then instantiate them based on different attention mechanisms.

A transformer can be seen as a composition of several sequence-to-sequence transformations. More precisely, a *seq-to-seq transformation* is a length-preserving $f : (\mathbb{R}^l)^* \to (\mathbb{R}^h)^*$ for some positive integers l, h. That is, f maps an input sequence σ of vectors of dimension l to an output sequence $f(\sigma)$ of dimension h of the same length, i.e., $|f(\sigma)| = |\sigma|$. We write $\texttt{iDim}(f)$ (resp. $\texttt{oDim}(f)$) to denote the dimension of the input (resp. output) vectors of f, i.e., l (resp. h). A sequence $\mu := f_1, \ldots, f_k$ of seq-to-seq transformers is said to be *well-typed* if $\texttt{iDim}(f_{i+1}) = \texttt{oDim}(f_i)$ for each $i = 1, \ldots, k-1$. We assume a finite *alphabet* Σ of tokens (a.k.a. symbols or characters) not containing the *end-of-string symbol* $\texttt{EOS}$. We write $\Sigma_{\texttt{EOS}}$ to denote $\Sigma \cup \{\texttt{EOS}\}$. A *transformer* $\mathcal{T}$ over Σ can then be defined as a triple $(\mu, \texttt{em}, \mathbf{t})$, where μ is a well-typed sequence of seq-to-seq transformers as above, $\texttt{em} : \Sigma_{\texttt{EOS}} \to \mathbb{R}^d$ with $d = \texttt{iDim}(f_1)$ is called a *token embedding*, and $\mathbf{t} \in \mathbb{R}^s$ with $s = \texttt{oDim}(f_k)$. The token embedding $\texttt{em}$ can be extended to $\texttt{em} : \Sigma^* \to (\mathbb{R}^d)^*$ by morphism, i.e., $\texttt{em}(w_1 \cdots w_n) = \texttt{em}(w_1) \cdots \texttt{em}(w_n)$, with $w_1 \cdots w_n \in \Sigma^*$. The language $L \subseteq \Sigma^*$ accepted by $\mathcal{T}$ consists precisely of strings $w \in \Sigma^*$ such that the last vector $\mathbf{v}$ in

$$f_k(f_{k-1}(\cdots f_1(\texttt{em}(w\texttt{EOS}))\cdots)) \tag{1}$$

— that is, at position $|w|+1$ in the sequence — satisfies $\langle \mathbf{t}, \mathbf{v} \rangle > 0$, where $\langle \mathbf{t}, \mathbf{v} \rangle$ denotes the dot product of $\mathbf{t}$ and $\mathbf{v}$. That is, we first apply $f_1, \ldots, f_k$ (in this order) to the sequence $\texttt{em}(w\texttt{EOS})$ of vectors, and check if a weighted sum of the arguments in the last vector is positive.

Remark 1. The above setting of transformers does not admit *Chain of Thoughts (CoTs)*. With CoTs, a transformer $\mathcal{T}$ on input w will output symbols, which are then continuously fed back into $\mathcal{T}$ until a specific output symbol is produced. That is, on input w, $\mathcal{T}$ produces a symbol a_1. We then run $\mathcal{T}$ on input wa_1 and produce another symbol a_2, and so on. It is known that transformers with CoTs

are Turing-complete [14, 3, 13]. In the sequel, we do not consider transformers with CoTs. □

We have thus far defined the notion of transformers only semantically. We now discuss how to define a seq-to-seq transformation more concretely. To this end, we employ the following ideas:

1. Use *piecewise linear functions* to modify a vector in the sequence.
2. Use *attention* to "aggregate" several vectors in the sequence.

We will discuss these in turn.

2.1 Piecewise linear functions

A *piecewise linear function* is a function $f : \mathbb{R}^r \to \mathbb{R}^s$ that is representable by a Feed-Forward Neural Network (FFNN). More precisely, a piecewise linear function can be defined inductively:

(Base) Each identity function $Id : \mathbb{R}^r \to \mathbb{R}^r$ is piecewise linear.
(Affine) If $f : \mathbb{R}^r \to \mathbb{R}^s$ is piecewise linear and $g : \mathbb{R}^s \to \mathbb{R}^t$ is an affine transformation[5], then the composition $f \circ g : \mathbb{R}^r \to \mathbb{R}^t$ is piecewise linear.
(ReLU) If $f : \mathbb{R}^r \to \mathbb{R}^s$ is piecewise and $i \in \{1, \ldots, s\}$, then the function $g : \mathbb{R}^r \to \mathbb{R}^s$ defined as

$$g(\mathbf{v}) = (w_1, \ldots, w_{i-1}, \max\{0, w_i\}, w_{i+1}, \ldots, w_s),$$

where $f(\mathbf{v}) = (w_1, \ldots, w_s)$, is piecewise linear.

As before, we can extend each piecewise linear function to sequences of vectors by morphisms, i.e., $f : (\mathbb{R}^r)^* \to (\mathbb{R}^s)^*$ with $f(\mathbf{v}_1, \ldots, \mathbf{v}_n) := f(\mathbf{v}_1), \ldots, f(\mathbf{v}_n)$. Notice, however, such functions can *only* modify a vector at the ith position in the sequence solely based on its values and *not* on the values of vectors at other positions. An intra-sequence aggregation of values is enabled by the so-called *attention*, which we discuss next.

2.2 Attention layers

To define an attention layer, we assume a *weight normalizer* $\texttt{wt} : \mathbb{R}^* \to \mathbb{R}^*$, which turns any d-sequence of weights into another d-sequence of weights. We will define some common normalizers below, which will result in hard and soft attention layers.

A seq-to-seq transformation $f : (\mathbb{R}^r)^* \to (\mathbb{R}^s)^*$ generated by an attention layer associated with $\texttt{wt}$ is given by three piecewise linear functions A, B, C

$$A, B : \mathbb{R}^r \to \mathbb{R}^r \qquad C : \mathbb{R}^{2r} \to \mathbb{R}^s.$$

[5] That is, given an input vector $\mathbf{x}$, we output $A\mathbf{x}+\mathbf{c}$, where A is a linear transformation and $\mathbf{c}$ is a constant vector.

defined as follows. On input $\sigma = \mathbf{x}_1, \ldots, \mathbf{x}_n$, we have $f(\sigma) = \mathbf{y}_1, \ldots, \mathbf{y}_n$ such that

$$\mathbf{y}_i := C(\mathbf{x}_i, \mathbf{v})$$

where

$$\mathbf{v} := \sum_{j=1}^{n} \mathbf{w}(j)\mathbf{x}_j, \tag{2}$$

$$\mathbf{w} := \mathtt{wt}(\{\langle A\mathbf{x}_i, B\mathbf{x}_j\rangle\}_{j=1}^{n}). \tag{3}$$

In other words, an attention layer looks at a vector $\mathbf{x}_i$ at each position i and decides "how much attention" is to be given to vectors $\{\mathbf{x}_j\}_{j=1}^{n}$ at any position in the input sequence. To this end, one obtains a sequence of weights $\{\langle \mathbf{x}_i, \mathbf{x}_j\rangle\}_{j=1}^{n}$. After normalizing this using $\mathtt{wt}$, the result of the attention is $\mathbf{v}$, which is a weighted sum $\{\mathbf{x}_j\}_{j=1}^{n}$ over all the input vectors.

Soft Attention. Practical transformers use weight normalizers defined by the softmax function, which turns a sequence of weights into a probability distribution. In particular, given a sequence $\sigma = x_1, \ldots, x_n \in \mathbb{R}^n$, define $\mathtt{softmax}(\sigma) := y_1, \ldots, y_n$, where

$$y_i := \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}.$$

A *SoftMax Attention Transformer (SMAT)* consists of seq-to-seq transformations that are defined using the softmax weight normalizer.

Hard Attention As previously mentioned, softmax attention is sometimes rather difficult to analyze, owing to the usage of exponential functions. This led researchers to use other weight normalizers that led to the so-called *hard attention layers*. More precise, there are two common flavors: *unique hard attention* and *average hard attention*. A unique hard attention uses the weight normalizer $\mathtt{uha}$ that selects the leftmost maximum weight, i.e., $\mathtt{uha}(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, where $y_i := 1$ if i is the leftmost position in $\mathbf{x} := x_1, \ldots, x_n$ with $x_i = \max(\mathbf{x})$; or else $y_i := 0$. An average hard attention uses the weight normalizer $\mathtt{aha}$ that selects *all* positions with maximum weight, i.e., $\mathtt{aha}(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, where $y_i := 1$ if $x_i = \max(\mathbf{x})$; or else $y_i := 0$.

2.3 Positional information

Thus far, we have actually defined a rather weak class of transformers (called *NoPE-transformers*) that cannot distinguish different positions in the input sequence. They recognize *permutation-invariant* languages, i.e., a string s is in the language L iff all of the reorderings of s are in L. There are two common ways to recover ordering: (1) *masking* and (2) *Position Embeddings (PEs)*. We will go through these in turn.

Masking. Masking is used to "hide" some positions in an input sequence to a layer with respect to a certain "anchor" position. The most commonly used type of masking is called *strict future masking*, which we will focus on in the remainder of the paper.

Intuitively, when attention is applied with respect to the position i, we looked at *all* positions and computed a normalized weight sequence accordingly. The version with strict future masking modifies this by considering only positions j *strictly before* i, i.e., $j < i$. Formally, one simply modifies Equation 2 and Equation 3 by the masked version:

$$\mathbf{v} = \sum_{j=1}^{i-1} \mathbf{w}(j)\mathbf{x}_j, \qquad \mathbf{w} = \mathtt{wt}(\{\langle A\mathbf{x}_i, B\mathbf{x}_j\rangle\}_{j=1}^{i-1}).$$

Position Embeddings (PEs). A *Position Embedding* is an *arbitrary* function of the form $p : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^d$. The idea is that $p(i, n)$ indicates the position information of the vector at position i for a sequence of length n. Thus, to extend transformers by PEs, we first apply both the token embedding and the PE p to the input string $w = w_1 \cdots w_n$ before processing the resulting sequence of vectors in the usual way. More formally, we modify the above acceptance condition in the definition of transformers by using

$$f_k(f_{k-1}(\cdots f_1(\sigma))\cdots))$$

where, instead of Equation 1, we use

$$\sigma := \mathtt{em}(w_1) + p(0, n+1), \cdots, \mathtt{em}(w_n) + p(n, n+1), \mathtt{em}(\mathtt{EOS}) + p(n+1, n+1).$$

At this point, it is appropriate to ask what types of PEs are reasonable. In practice, PEs may use trigonometric functions (e.g. sin) and various other information about the position in the sequence (e.g. the "absolute" position i, the length n of the sequence, etc.). Thus, researchers have studied transformers with PEs *without* any restriction whatsoever on the PEs. Remarkably, some interesting results can already be proven in this setting. We will mention some restricted classes later.

We state a result that can be easily proven:

Proposition 1. *Each Masked UHAT (resp. AHAT) with PEs can be simulated by UHAT (resp. AHAT) with PEs with no masking.*

3 Unique Hard Attention Transformers

The first fundamental result concerning UHAT comes from [5, 6], which show that their class of languages is contained in the well-studied circuit complexity class AC^0, consisting of problems solvable by constant-depth, polynomial-size Boolean circuits. More recently, [2] proved that this containment is strict.

Theorem 1 ([5, 6, 2]). *UHAT with PEs is strictly subsumed in AC^0.*

Proof idea. Let us quickly discuss the proof idea behind the containment in AC^0. Fix an UHAT $\mathcal{T}$ with, say, h layers. For simplicity, let us assume the alphabet $\Sigma = \{a, b\}$. The key idea is that there is a polynomial function $p(n)$, for any possible string length n, such that the set V_n of vectors — as well as the set S_n of possible attention scores — that can be generated in the computation of the UHAT has size $|V_n| = O(p(n))$. More precisely, in the input layer after application of `em` and position encoding, we can generate $O(n)$ many vectors. In the next layer, there are $O(n^2)$ many vectors. In the kth layer, there are $O(n^{2^{k-1}})$ possible vectors. Therefore, we may set $p(n)$ to be $O(n^{2^h})$.

Thus, we may represent each vector in V_n and each attention score in S_n using $O(\log n)$ bits. Therefore, using a constant depth polynomial-sized boolean circuit (by a simple enumeration), we can represent the relation $\preceq \subseteq S_n \times S_n$ containing pairs (s, s') such that s has a smaller attention score as s'. Similarly, using a constant depth polynomial-sized boolean circuit, we can represent the relation $R \subseteq V_n \times V_n \times S_n$ such that $R(\mathbf{v}, \mathbf{v}', s)$ iff $\langle \mathbf{v}, \mathbf{v}' \rangle = s$. Together, this allows us to represent — using constant depth polynomial-sized boolean circuits — the function $f_\ell : V_n^n \times \{1, \ldots, n\} \to V_n$ such that $f_\ell((\mathbf{v}_1 \cdots \mathbf{v}_n), i)) = \mathbf{v}$ iff, whenever the ℓth layer has input sequence $\mathbf{v}_1 \cdots \mathbf{v}_n$, the vector at position i at layer $\#(\ell+1)$ is $\mathbf{v}$. All in all, this gives rise to a constant-depth polynomial-sized boolean circuit C_n for input strings of length n.

To conclude the theorem, we simply use the (non-uniform) family $\{C_n\}_{n \geq 0}$ of circuits to represent $\mathcal{T}$. □

Combined with well-known limitations of AC^0 (e.g. see [1, 11]), the above result shows that some languages are not expressible by UHAT, including PARITY and MAJ, where the latter is defined as:

$$\mathrm{MAJ} := \{w \in \{a, b\}^* : |w|_a \geq |w|_b\}.$$

While this provides us a ceiling of what languages are expressible as UHATs, the following two results show what UHATs are capable of. To this end, we write FO[Mon] to denote first-order logic over strings extended only by *monadic* numerical predicates (i.e. sets of numbers); recall that this would have yielded AC^0 if extended with all k-ary ($k \geq 1$) numerical predicates; see [11]. An example of monadic numerical predicates is Mod_2^3 containing all numbers that are 2 (mod 3).

Theorem 2 ([2, 17]). *FO[Mon] is expressible by UHAT with PEs, as well as by masked UHATs with finite-image PEs. In addition, masked NoPE-UHAT coincides with FO, which in turn coincides with star-free languages.*

Proof idea. To prove the containment of FO[Mon] in UHAT —either with PEs or masked attention with finite image PEs— we use Kamp's theorem [8]: FO[Mon] coincides in expressivity with LTL[Mon], i.e., LTL formulas that also use monadic numerical predicates as atomic propositions. Unlike FO formulas, which have

multiple variables, LTL formulas are *unary*, meaning that their semantics is a set of positions over a string. This simpler structure of LTL aligns well with the expressive power of UHATs, allowing for a proof using structural induction. In particular, we inductively show that for every LTL formula ϕ with unary numerical predicates, there exists a UHAT $\mathcal{T}_\phi$ such that on input $\sigma = \mathbf{x}_1, \ldots, \mathbf{x}_n$, corresponding to the embedding of a word $w = a_1, \ldots, a_n \in \Sigma^+$, it outputs a sequence $\mathcal{T}_\phi(\sigma) = \mathbf{y}_1, \ldots, \mathbf{y}_n$ over $\{0, 1\}$ that contains a 1 precisely in those positions of w that satisfy ϕ.

Let us give some intuition on how to do the aforementioned induction proof. For the base case, we deal with only Q_a (saying that the current position has letter a) or a monadic numerical predicate $U \subseteq \mathbb{N}$. We need to set up the token embedding function `em` and position embedding p with a large enough dimension so that information on truth/falsehood of each atomic proposition in the given LTL formula can be read off directly. For example, for a string $w := abaa$ with the LTL formula $\mathbf{G}(\mathrm{Mod}_2^2 \to Q_a)$, we would map w to the following sequence of vectors:

$$(1, 0, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1)$$

where the vector at position i corresponds to $(Q_a(i), Q_b(i), \mathrm{Mod}_2^2(i))$. Note, we omitted `EOS` and potentially other "information" in the PEs for readability.

For the inductive case, one introduces new arguments at each position (i.e. increases the dimension) to encode truth/falsehood of the formulas higher up in the parse tree. Note, we keep the information stored in the previous layer.

For boolean combinations, one can handle this with piecewise linear functions. That is, $\neg\varphi$ can be implemented by the function $1 - x_\varphi$, where x_φ encodes the value of φ at the same position in the string. For $\varphi \vee \psi$, we can implement it as $x_\varphi + \mathrm{ReLU}(x_\psi - x_\varphi) = x_\varphi + \max(0, x_\psi - x_\varphi)$.

We next give an intuition how to do $\mathbf{F}\varphi$ and show how to do this with PEs (with no masking). For other temporal operators, the reader is referred to [2]. To this end, we assume by induction that the value x_φ and $x_{\neg\varphi}$ are available at every position in the sequence. The first step is to "nullify" the value $x_{\neg\varphi}$ at the last position n, i.e., $x_{\neg\varphi}[n] := 0$. See the proof of Lemma 1 in [2]. We then assume the use the following information at position i:

$$\mathbf{v}_i := \langle \cos(\pi(1 - 2^{-i})/10), \sin(\pi(1 - 2^{-i})/10), 1, x_{\neg\varphi} \rangle.$$

With an appropriate affine transformation B, we have

$$B\mathbf{v}_i := \langle \cos(\pi(1 - 2^{-i})/10), \sin(\pi(1 - 2^{-i})/10), -10.x_{\neg\varphi}, 0 \rangle.$$

Thus, we have

$$\langle \mathbf{v}_i, B\mathbf{v}_j := \cos(\pi(2^{-i} - 2^{-j})/10) - 10.x_{\neg\varphi}.$$

The value $\cos(\pi(2^{-i} - 2^{-j})/10)$ is maximized at position $j \geq i$ and not at $j < i$. In addition, the value $-10.x_{\neg\varphi}$ is maximized at $j = n$ (possibly also at $j < i$). Thus, it follows that $\langle \mathbf{v}_i, B\mathbf{v}_j \rangle$ is maximized at position $j \geq i$. Furthermore, it can be verified that among the value $j \geq i$ the value $\cos(\pi(2^{-i} - 2^{-j})/10)$

monotonically decreases in j. All in all, unique hard attention picks the vector $\mathbf{v}$ at the leftmost position $j \geq i$ such that $w, j \models \varphi$ (otherwise, it picks the vector $\mathbf{v}$ at position n), with which we can forward the truth/falsehood of $\mathbf{F}\varphi$. □

Corollary 1. *UHAT with PEs contain all regular languages expressible in AC^0.*

Proof. It is known that all regular languages in AC^0 are expressible in FO with unary numerical predicates (more precisely,Mod_r^d containing all numbers that are in the same equivalence class as $r \pmod d$). The corollary then follows from Theorem 2. □

Theorem 2 turns out to be powerful enough to show the following interesting "non-regular" capability of UHAT with PEs.

Corollary 2 ([2]). *Palindrome is in UHAT with PEs.*

Proof idea. Using PEs, it is possible to extend Theorem 2 with any desired family $\{\preceq_n\}_{n\geq 0}$, where $\preceq_n$ deals with strings of length n. Therefore, on strings of length n, we could use the ordering

$$1, n, 2, n-2, 3, n-3, \ldots$$

of the set $\{1, \ldots, n\}$. This essentially turns the string *abccba* into *aabbcc*, for example. Therefore, using the unary numerical predicate Mod_1^2, we can write an LTL[Mon] (or equivalent FO[Mon]) formula that says that at each odd position i the next position $i+1$ has to have the same label as that at position i. □

We conclude our discussion of UHAT by the problem of verifying Masked UHAT with no PEs. By verifying, this could mean checking the emptiness, universality of the language, or its equivalence to (or containment in) another Masked UHAT. By Theorem 2, each Masked UHAT can be effectively turned into a finite-state automaton recognizing the same language. Owing to decidability of emptiness, universality, equivalence, and containment for finite automata, we obtain the same decidability results for Masked UHAT with no PEs.

Corollary 3 ([17]). *The problem of verifying Masked UHAT with no PEs is decidable.*

4 Logical Languages for Average Hard Attention

It is easy to construct an AHAT that recognizes MAJ. This takes AHAT beyond AC^0. The following result shows that TC^0 still upper-bounds the capability of AHAT.

Theorem 3 ([6]). *Languages recognized by AHAT are in TC^0*

The main reason behind the TC^0 upper bound of AHAT is the ability of TC^0-circuits to simulate arithmetic, which is needed in the computation of average hard attention.

For the time being no complete characterization for neither AHAT with PEs nor masked NoPE-AHAT exists. That is, we do not have an extension of Theorem 2 to AHAT. However, it is still possible to specify a logic that expresses languages that can be expressed by AHATs. The logic is called *Counting LTL*, as first defined in [2]. Intuitively, Counting LTL extends LTL with linear counting terms of the form:

$$C, C' \;:=\; c \; (c \in \mathbb{Z}) \mid \overleftarrow{\#}[\varphi] \mid \overrightarrow{\#}[\varphi] \mid C + C' \mid C - C',$$

and formulas of the form $C \leq C'$, where C and C' are linear counting terms. The term $\overleftarrow{\#}[\varphi]$ (resp. $\overrightarrow{\#}[\varphi]$) counts the number of times φ holds at positions before (resp. after) the one where we are evaluating the formula. The remaining terms and formulas have an intuitive meaning.

We define the fragment $\mathrm{K}_t[\#]$ of the Counting LTL, which removes all temporal operators of LTL, as well as terms of the form $\overrightarrow{\#}[\psi]$. That is, only terms of the form $\overleftarrow{\#}[\varphi]$ is allowed. For instance, if Q_a and Q_b are formulas that check whether a position in a word holds symbol a or b, respectively, then the $\mathrm{K}_t[\#]$ formula $\overleftarrow{\#}[Q_b] \leq \overleftarrow{\#}[Q_a]$ checks whether the word belongs to MAJ (if evaluated on the last position of the word). Similarly, we can define Dyck-1, the language of well-matched parenthesis words over the alphabet consisting of tokens (and). The $\mathrm{K}_t[\#]$ that checks for this language over the last position of a word in this alphabet is:

$$\overleftarrow{\#}[Q_(] = \overleftarrow{\#}[Q_)] \wedge \overleftarrow{\#}[\overleftarrow{\#}[Q_)] > \overleftarrow{\#}[Q_(]] = 0,$$

where we have used some standard logical abbreviations. It is possible to show that the Counting LTL can express PARITY, whereas $\mathrm{K}_t[\#]$ cannot express PARITY [7].

Theorem 4 ([2, 16]). *Counting LTL extended with unary numerical predicates is in AHAT with PEs. The fragment $K_t[\#]$ is expressible by masked NoPE-AHAT.*

The basic idea behind the proofs of these results is that AHATs allow to compute the uniform average value among all positions that maximize the attention. This averaging mechanism allows to express many counting properties of interest. The proof is, again, by structural induction on Counting LTL formulas.

We showed that UHAT contains all regular languages in AC^0. We do not know if this is true for AHAT. That said, Theorem 4 can be used to show the following slightly weaker result:

Corollary 4 ([2]). *If TC^0 is strictly contained in NC^1, then AHAT with PEs contains all regular languages in TC^0.*

It turns out that, for the subclass of AHATs with no masking and no PEs, the following complete characterization can be proven:

Theorem 5 ([9]). *NoPE-AHAT recognizes precisely all permutation-invariant languages with semi-algebraic Parikh images.*

To explain this theorem, recall (see [10]) that the Parikh image $\mathcal{P}$ of a language is a mapping from all strings in the language to their letter counts. For example, $\mathcal{P}((ab)^*) = \{(n, n) : n \in \mathbb{N}\}$. Here, the tuple $(3, 3)$ simply denotes that there are 3 occurrences of as and 3 occurrences of b's. Parikh's Theorem shows that context-free languages have semilinear Parikh images, i.e., they are definable in Presburger Arithmetic. In contrast, a relation $R \subseteq \mathbb{N}^k$ is *semi-algebraic* if it can be expressed as a finite union of nonnegative integer solutions to systems of multivariate polynomial inequations. That is, the above theorem implies (among others) that languages L_k of the form $\{w \in \{a, b\}^* : |w|_a \geq (|w|_b)^k$, are expressible by NoPE-AHAT; note that L_k has no semilinear Parikh images for $k \geq 2$. Interestingly, this also shows that Counting LTL does not subsume NoPE-AHAT, since the former can only express permutation-invariant languages with semilinear Parikh images [9].

Theorem 5 yields immediately undecidability of verification of NoPE-AHAT since solvability of Diophantine equations is well-known to be undecidable [12].

Corollary 5 ([9]). *Checking whether a NoPE-AHAT recognizes a nonempty language is undecidable.*

5 Limitations of UHATs and AHATs

Having gone through some body of results in the literature, we now discuss two main limitations of these results.

Limitation 1: Soft attention vs. Hard attention. As we remarked, practical transformers are based on soft attention. It is still unclear whether the theory of expressivity of UHATs and AHATs provides a good approximation of the theory of expressivity of softmax transformers. For example, we do not know where the expressivity of softmax transformers exactly lies (e.g. do they subsume UHATs?). That said, it is known that PARITY can be captured by a softmax transformer. Thus, softmax transformers are not subsumed by UHATs [4]. Furthermore, the relationship between AHAT and softmax transformers has also not been fully delineated (for more on this, see [18, 16]).

One subclass of AHAT that seems to be a promising approximation of SMAT restricts all layers to apply only *uniform* attention. More precisely, an AHAT layer is uniform if the piecewise linear functions $A, B : \mathbb{R}^r \to \mathbb{R}^r$ ensure that there exists a constant c such that $\langle A\mathbf{x}, B\mathbf{y}\rangle = c$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^r$. This can happen esp. when the linear transformation components of A and B map $\mathbf{x}$ and $\mathbf{y}$ to 0. The subclass is denoted by AHAT[U]. The following result is folklore and can easily be shown by noting that $\texttt{softmax}(s_1, \ldots, s_n) = \texttt{aha}(s_1, \ldots, s_n) = 1/n$, whenever $s_1 = \cdots = s_n$, which can be guaranteed for uniform AHAT layers.

Proposition 2. *Language recognized by AHAT[U] are also recognized by SMAT.*

The above observation was already used in [9, 16] to show the power of SMAT:

Proposition 3 ([16]). *$K_t[\#]$ languages are recognizable by SMAT.*

Proposition 4 ([9]). *Permutation-invariant languages with semialgebraic Parikh images are recognizable by SMAT.*

Limitation 2: Trainability vs. expressibility. Not all expressible languages are efficiently trainable on transformers, i.e., by means of Stochastic Gradient Descent (SGD). This applies particularly to PARITY [4], which seems to be extremely difficult to train on transformers with any high enough level of accuracy, although it is expressible by a softmax transformer. This phenomenon was very recently shown to be caused by *sensitivity*. Loosely speaking, PARITY is sensitive since flipping one letter (i.e. a to b and vice versa) changes the parity of any string. Contrast this with MAJ, where there are not so many strings that change their memberships in MAJ, after flipping a letter. This was hypothesized to be the reason why MAJ is efficiently trainable, whereas PARITY is not.

One interesting upshot of the research effort in understanding trainability is the so-called *RASP-L conjecture* [19], which states that a concept is likely to length generalize (i.e. when trained on shorter strings, generalize to longer strings) precisely whenever it is expressible as a short RASP-L program. However, as noted by Huang et al. [7], this is not a precisely formulated conjecture. The authors postulated a formal version of RASP-L conjecture by replacing RASP-L with the logic $\mathrm{K}_t[\#]$, for which they could successfully prove and empirically verify a length generalization theorem. In particular, this ruled out PARITY (as it is not in $\mathrm{K}_t[\#]$), but admits MAJ. It remains to be seen if $\mathrm{K}_t[\#]$ subsumes all concepts that admit length generalization on transformers.

6 Conclusions

We have discussed several key results employing logic and automata for understanding what is expressible in/efficiently trainable for transformers. It must be emphasized that these are only a handful of results in this rapidly growing field of FLaNN (Formal Languages and Neural Networks); for a more comprehensive (though less detailed) account of FLaNN, see the excellent survey [15]. It is our sincere hope that this article could motivate more researchers in logic and automata, as well as verification and programming languages, to take up some of the many pressing challenges in FLaNN.

Acknowledgment. We thank David Chiang, Michael Hahn, Alexander Kozachinskiy, Andy Yang, and Georg Zetzsche for the fruitful discussion. Lin is supported by the European Research Council[6] under Grant No. 101089343 (LASD). Barceló is funded by ANID - Millennium Science Initiative Program - Code ICN17002, and by CENIA FB210017, Basal ANID.

[6] https://doi.org/10.13039/100010663

References

1. Ajtai, M.: $\sum^1_1$-formulae on finite structures. Ann. Pure Appl. Log. **24**(1), 1–48 (1983)
2. Barceló, P., Kozachinskiy, A., Lin, A.W., Podolskii, V.V.: Logical languages accepted by transformer encoders with hard attention. In: ICLR (2024)
3. Bhattamishra, S., Patel, A., Goyal, N.: On the computational power of transformers and its implications in sequence modeling. In: CoNLL. pp. 455–475 (2020)
4. Chiang, D., Cholak, P.: Overcoming a theoretical limitation of self-attention. In: Muresan, S., Nakov, P., Villavicencio, A. (eds.) ACL. pp. 7654–7664 (2022)
5. Hahn, M.: Theoretical limitations of self-attention in neural sequence models. Trans. Assoc. Comput. Linguistics **8**, 156–171 (2020)
6. Hao, Y., Angluin, D., Frank, R.: Formal language recognition by hard attention transformers: Perspectives from circuit complexity. Trans. Assoc. Comput. Linguistics **10**, 800–810 (2022)
7. Huang, X., Yang, A., Bhattamishra, S., Sarrof, Y.R., Krebs, A., Zhou, H., Nakkiran, P., Hahn, M.: A formal framework for understanding length generalization in transformers. In: ICLR (2025)
8. Kamp, H.W.: Tense Logic and the Theory of Linear Order. Ph.D. thesis, University of California, Los Angeles (1968)
9. Köcher, C., Kozachinskiy, A., Lin, A.W., Sälzer, M., Zetzsche, G.: Nope: The counting power of transformers with no positional encodings. CoRR **abs/2505.11199** (2025)
10. Kopczynski, E., To, A.W.: Parikh images of grammars: Complexity and applications. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom. pp. 80–89 (2010). https://doi.org/10.1109/LICS.2010.21, https://doi.org/10.1109/LICS.2010.21
11. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
12. Matiyasevich, Y.V.: Hilbert's Tenth Problem. MIT Press, Cambridge, Massachusetts (1993)
13. Merrill, W., Sabharwal, A.: The expressive power of transformers with chain of thought. In: ICLR (2024)
14. Pérez, J., Barceló, P., Marinkovic, J.: Attention is turing-complete. J. Mach. Learn. Res. **22**, 75:1–75:35 (2021)
15. Strobl, L., Merrill, W., Weiss, G., Chiang, D., Angluin, D.: What formal languages can transformers express? A survey. Trans. Assoc. Comput. Linguistics **12**, 543–561 (2024)
16. Yang, A., Chiang, D.: Counting like transformers: Compiling temporal counting logic into softmax transformers. CoRR **abs/2404.04393** (2024)
17. Yang, A., Chiang, D., Angluin, D.: Masked hard-attention transformers recognize exactly the star-free languages. In: NeurIPS (2024)
18. Yang, A., Strobl, L., Chiang, D., Angluin, D.: Simulating hard attention using soft attention. CoRR **abs/2412.09925** (2024)
19. Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J.M., Bengio, S., Nakkiran, P.: What algorithms can transformers learn? A study in length generalization. In: ICLR (2024)