Tighter Bounds on the Expressivity of Transformer Encoders

David Chiang [1] **Peter Cholak** [1] **Anand Pillay** [1]

Abstract

Characterizing neural networks in terms of better-understood formal systems has the potential to yield new insights into the power and limitations of these networks. Doing so for transformers remains an active area of research. Bhattamishra and others have shown that transformer encoders are at least as expressive as a certain kind of counter machine, while Merrill and Sabharwal have shown that fixed-precision transformer encoders recognize only languages in uniform TC^0. We connect and strengthen these results by identifying a variant of first-order logic with counting quantifiers that is simultaneously an upper bound for fixed-precision transformer encoders and a lower bound for transformer encoders. This brings us much closer than before to an exact characterization of the languages that transformer encoders recognize.

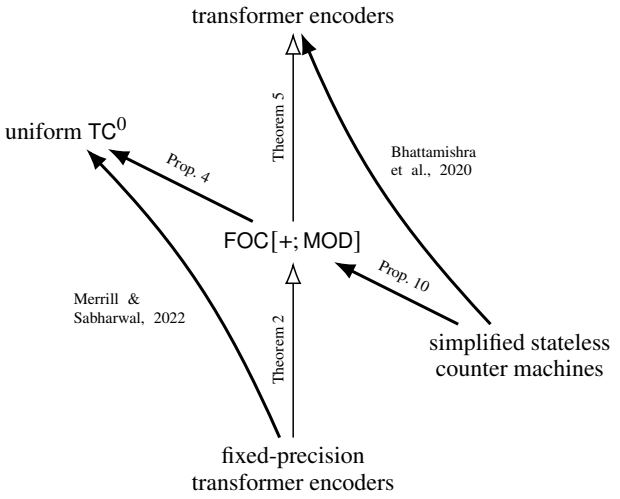


Figure 1. Overview of results. Arrows indicate inclusion, and thick arrows indicate strict inclusion. We show that $\mathsf{FOC}[+;\mathsf{MOD}]$ is simultaneously a tighter upper bound on fixed-precision transformer encoders than uniform TC^0 is, and a tighter lower bound on transformer encoders than SSCMs are.

1. Introduction

Characterizing neural networks in terms of better-understood formal systems has the potential to yield new insights into the power and limitations of these networks. Recurrent neural networks (RNNs) were linked to finite automata from the start (McCulloch & Pitts, 1943; Kleene, 1956) and have continued to be studied using finite automata (see, e.g., the survey by Forcada & Carrasco (2001)). Convolutional neural networks, too, have been related to finite automata (Schwartz et al., 2018).

Transformers (Vaswani et al., 2017) have been studied in relation to counter machines (Bhattamishra et al., 2020), Boolean circuits (Hao et al., 2022; Merrill et al., 2022; Merrill & Sabharwal, 2023), and programming languages (Weiss et al., 2021), obtaining various upper and lower bounds on their expressivity. (Section 7 gives a more detailed survey.) As a lower bound, Bhattamishra et al. (2020) show that transformer encoders are at least as powerful as *simplified stateless counter machines* (SSCMs), which test whether the numbers of occurrences of input symbols satisfy a given linear constraint. As an upper bound, Merrill & Sabharwal (2022) restrict to *fixed-precision* transformer encoders and show that they are in uniform TC^0.

Here, we study transformers in relation to first-order logic. (Merrill & Sabharwal (2022) also relate them to first-order logic, but indirectly through circuits.) We define $\mathsf{FOC}[+;\mathsf{MOD}]$, which is first-order logic with counting quantifiers, where positions have modular predicates but not ordering, and counts have ordering and addition (§4). We then connect and strengthen the two above-mentioned results by showing that $\mathsf{FOC}[+;\mathsf{MOD}]$ is simultaneously an upper bound for fixed-precision transformer encoders (§5) and a lower bound for transformer encoders (§6).

These two results counterbalance each other. As an upper bound on fixed-precision transformer encoders, $\mathsf{FOC}[+;\mathsf{MOD}]$ might be imagined to define languages far beyond the power of transformers, but the lower bound assures us that it does not have any expressivity that does any-

[1]University of Notre Dame, USA. Correspondence to: David Chiang <dchiang@nd.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

thing "un-transformer-like." As a lower bound on (arbitrary-precision) transformer encoders, $\mathsf{FOC}[+;\mathsf{MOD}]$ might be imagined to be far weaker than transformers, but the upper bound assures us that it can express everything that real-world (fixed-precision) transformers can. Together, these two results bring us much closer than before to an exact characterization of the languages that transformer encoders recognize.

2. Preliminaries

We write $\mathbb{N}$ for the set of natural numbers, which includes 0. If a and b are integers, we write $[a, b]$ for the set $\{x \in \mathbb{Z} \mid a \le x \le b\}$, and $a \equiv_m b$ iff a and b are congruent modulo m.

We make frequent use of the Iverson bracket $\mathbb{I}[\phi]$, which has the value 1 if the statement ϕ is true, and 0 otherwise.

If M is a matrix, we write $M_{i,*}$ for the i-th row of M and $M_{*,j}$ for the j-th column of M. We write $\mathbf{0}$ for the zero vector or matrix, $\mathbf{0}^n$ for the n-dimensional zero vector, and $\mathbf{I}$ for the identity matrix.

We often work with families of functions $f = \left(f^{(n)}\right)_{n>0}$ where each $f^{(n)} \colon X^n \to Y^n$. For brevity, we write $f \colon X^n \to Y^n$ and apply f to strings or vectors of any length.

3. Transformers

A transformer can have an encoder and/or a decoder; following previous work (Bhattamishra et al., 2020; Hahn, 2020; Hao et al., 2022; Merrill et al., 2022; Merrill & Sabharwal, 2023), we focus on transformer encoders.

The input is a string $w_1 \cdots w_n$, to which we prepend a special symbol $\mathtt{CLS}$ at position 0. Thus the network sees a sequence of $n+1$ symbols; to avoid clutter, we write $n' = n+1$. The string is converted to an *activation* matrix $A^{(0)} \in \mathbb{R}^{d \times n'}$, where column $A^{(0)}_{*,p}$ represents symbol w_p, and d is the *width* of the network. A stack of L layers maps $A^{(0)}$ to $A^{(1)}, A^{(2)}, \ldots, A^{(L)} \in \mathbb{R}^{d \times n'}$. Then we apply a sigmoid layer to the output at $\mathtt{CLS}$ (that is, $A^{(L)}_{*,0}$) to obtain a single probability. The network, which we call a *transformer classifier*, accepts w iff this probability is at least $\frac{1}{2}$.

The rest of this section describes each of these components in more detail. Readers already familiar with transformers may safely skip ahead.

3.1. Input layer

Each input vector $A^{(0)}_{*,p}$ is the sum of a *word embedding* and a *positional encoding*.

Definition 1. A *word embedding* on Σ with width d is a mapping $\mathrm{WE} \colon \Sigma \to \mathbb{R}^d$ from symbols to vectors.

Definition 2. A *sinusoidal positional encoding* with (even) width d is a mapping from positions to vectors,

$$\mathrm{PE} \colon \mathbb{N} \to \mathbb{R}^d$$
$$p \mapsto \begin{bmatrix} \sin 2\pi\xi_1 p \\ \cos 2\pi\xi_1 p \\ \vdots \\ \sin 2\pi\xi_{d/2} p \\ \cos 2\pi\xi_{d/2} p \end{bmatrix} \qquad \xi_i \in \mathbb{Q}$$

In the original paper, the ξ_i are set to fixed values ($\xi_i = \frac{i-1}{2\pi}$), which are not rational. Here, we assume that they are rational (needed for Theorem 2), and can be set to arbitrary rationals (needed for Theorem 5). Since the rationals are dense in the reals, we can choose them to be as close to the original values as we want.

3.2. Hidden layers

Each hidden layer has a *self-attention* followed by a *position-wise feed-forward network*.

Definition 3. A *self-attention sublayer* with width d and key width d_{K} is a function

$$\mathrm{SA} \colon \mathbb{R}^{d \times n'} \to \mathbb{R}^{d \times n'}$$
$$A \mapsto \begin{bmatrix} c_0 & \cdots & c_n \end{bmatrix} \text{ where}$$
$$s_{qp} = \frac{W^{(\mathrm{Q})} A_{*,q} \cdot W^{(\mathrm{K})} A_{*,p}}{\sqrt{d}} \tag{1}$$
$$c_q = \frac{\sum_{p=0}^{n} (\exp s_{qp}) W^{(\mathrm{V})} A_{*,p}}{\sum_{p=0}^{n} \exp s_{qp}} \tag{2}$$

and $W^{(\mathrm{Q})}, W^{(\mathrm{K})} \in \mathbb{R}^{d \times d_{\mathrm{K}}}$ and $W^{(\mathrm{V})} \in \mathbb{R}^{d \times d}$ are learned.

Definition 4. A *position-wise feed-forward network* (FFN) with width d and hidden width d_{FF} is a function

$$\mathrm{FF} \colon \mathbb{R}^d \to \mathbb{R}^d$$
$$x \mapsto W^{(2)} \left(\max(0, W^{(1)} x + b^{(1)})\right) + b^{(2)}$$

where the $\max(0, -)$, called a *ReLU*, is taken elementwise, and $W^{(1)} \in \mathbb{R}^{d_{\mathrm{FF}} \times d}$, $b^{(1)} \in \mathbb{R}^{d_{\mathrm{FF}}}$, $W^{(2)} \in \mathbb{R}^{d \times d_{\mathrm{FF}}}$, $b^{(2)} \in \mathbb{R}^d$ are learned.

We also apply f column-wise to matrices $A \in \mathbb{R}^{d \times n'}$:

$$\mathrm{FF}(A) = \begin{bmatrix} \mathrm{FF}(A_{*,0}) & \cdots & \mathrm{FF}(A_{*,n}) \end{bmatrix}. \tag{3}$$

Definition 5. A *transformer layer* with H heads and width d is a function

$$\mathrm{Layer} \colon \mathbb{R}^{d \times n'} \to \mathbb{R}^{d \times n'}$$
$$A \mapsto A'' \text{ where}$$
$$A' = \sum_{h=1}^{H} \mathrm{SA}^{(h)}(A) + A$$
$$A'' = \mathrm{FF}(A') + A'$$

where the $\mathrm{SA}^{(h)}$ are self-attentions, and FF is a position-wise FFN. The $+A$ and $+A'$ terms are known as *residual connections*.

We have omitted layer normalization (Ba et al., 2016) to simplify proofs. Appendix D explains how to add layer normalization to our definitions and proofs.

3.3. Stacks, encoders and classifiers

Definition 6. A *transformer stack* with width d is a function

$$\mathrm{Stack}\colon \mathbb{R}^{d\times n'} \to \mathbb{R}^{d\times n'}$$
$$\mathrm{Stack} = \mathrm{Layer}^{(L)} \circ \cdots \circ \mathrm{Layer}^{(1)}$$

where each $\mathrm{Layer}^{(\ell)}$ is a transformer layer with width d.

Definition 7. A *transformer encoder* with width d is a function from strings to sequences of vectors,

$$\begin{aligned}\mathrm{Enc}\colon \Sigma^n &\to \mathbb{R}^{d\times n'}\\ w &\mapsto \mathrm{Stack}(A) \text{ where}\\ A &= \begin{bmatrix}\mathrm{WE}(\mathsf{CLS}) & \mathrm{WE}(w_1) & \cdots & \mathrm{WE}(w_n)\end{bmatrix}\\ &\quad + \begin{bmatrix}\mathrm{PE}(0) & \mathrm{PE}(1) & \cdots & \mathrm{PE}(n)\end{bmatrix}\end{aligned}$$

where WE is a word embedding, PE is a positional encoding, and Stack is a transformer stack with width d.

Definition 8. A *transformer classifier* with width d is a function

$$\begin{aligned}\mathrm{Cls}\colon \Sigma^* &\to \mathbb{R}\\ w &\mapsto \mathrm{sigmoid}\left(W\,[\mathrm{Enc}(w)]_{*,0} + b\right)\end{aligned} \tag{4}$$

where Enc is a transformer encoder with width d. We say that Cls *accepts* w if $\mathrm{Cls}(w) \geq \frac{1}{2}$, and the language recognized by Cls is $\{w \in \Sigma^* \mid \mathrm{Cls} \text{ accepts } w\}$.

4. First-Order Logic with Counting Quantifiers

Instead of characterizing problems (formal languages) using devices for producing or consuming strings, we can characterize them using *logical formulas* that declare what properties a string must have. The classic result in this approach is that the languages of finite strings described by monadic second-order logic are exactly those recognized by finite automata (Büchi, 1960). Merrill & Sabharwal (2022) relate the expressivity of transformers to that of first-order logic with majority quantifiers, but only indirectly via circuits. Here, we relate the expressivity of transformers directly to a logic called $\mathsf{FOC}[+;\mathsf{MOD}]$.

4.1. Examples

We begin with a few examples of sentences of $\mathsf{FOC}[+;\mathsf{MOD}]$ that define languages. Assume $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

1. $\forall p.Q_{\mathtt{0}}(p) \vee \forall p.Q_{\mathtt{1}}(p)$ defines the language $\mathtt{0}^* \cup \mathtt{1}^*$. The variable p ranges over positions of w, and for any symbol $a \in \Sigma$, $Q_a(p)$ is true iff the symbol at position p is a. So this sentence says that all symbols are $\mathtt{0}$ or all are $\mathtt{1}$.

2. $\forall p.(\mathsf{MOD}^0_2(p) \to Q_{\mathtt{0}}(p) \wedge \mathsf{MOD}^1_2(p) \to Q_{\mathtt{1}}(p))$ defines the language $(\mathtt{10})^* \cup (\mathtt{10})^*\mathtt{1}$. For any $r \geq 0, m > 0$, the predicate $\mathsf{MOD}^r_m(p)$ tests whether $p \equiv_m r$. So this sentence says that all symbols in even positions are $\mathtt{0}$'s and those in odd positions are $\mathtt{1}$'s.

3. $\exists x.\,(\exists^{=x} p.Q_{\mathtt{0}}(p) \wedge \exists^{=x} p.Q_{\mathtt{1}}(p))$ defines the language of strings with an equal number of $\mathtt{0}$'s and $\mathtt{1}$'s. The variable x ranges over numbers. The $\exists^{=x}$ is a *counting quantifier*; the subformula $\exists^{=x} p.Q_{\mathtt{0}}(p)$ says that there are exactly x positions p that make $Q_{\mathtt{0}}(p)$ true. Similarly, $\exists^{=x} p.Q_{\mathtt{1}}(p)$ says that there are exactly x occurrences of $\mathtt{1}$.

4. $\exists x.\exists y.\,(\exists^{=x} p.Q_{\mathtt{0}}(p) \wedge \exists^{=y} p.Q_{\mathtt{1}}(p) \wedge 2x = y)$ defines the language of strings with twice as many $\mathtt{1}$'s as $\mathtt{0}$'s. We allow linear equations or inequalities like $2x = y$, but only on count variables.

4.2. Definition

We now describe the syntax of $\mathsf{FOC}[+;\mathsf{MOD}]$, given a fixed finite alphabet Σ, and its intended interpretation with reference to finite strings $w = w_1 \cdots w_n$ over Σ. The syntax has two sorts (Immerman, 1999, p. 185–187):

- The sort of *positions* has variables $p, \ldots$, which stand for positions of w, that is, integers in $[1, n]$.
- The sort of *counts* has
  - variables $x, y, z, \ldots$, which stand for rational numbers
  - terms $c_0 + c_1 x_1 + \cdots + c_k x_k$, where each c_i is a rational number and each x_i is a count variable.

A formula of $\mathsf{FOC}[+;\mathsf{MOD}]$ is one of:

- $\top$ for true or $\bot$ for false.
- $Q_a(p)$ where $a \in \Sigma$, which is true iff $w_p = a$.
- $\mathsf{MOD}^r_m(p)$ where $r \geq 0, m > 0$, which is true iff $p \equiv_m r$.
- $t_1 = t_2$, $t_1 < t_2$, where t_1 and t_2 are terms (in the sort of counts).
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg\phi_1$ where ϕ_1 and ϕ_2 are formulas.
- $\exists x.\phi$, $\forall x.\phi$ where x is a count variable and ϕ is a formula.

- $\exists^{=x} p.\phi$, where x is a count variable, p is a position variable, and ϕ is a formula, which is true iff ϕ is true for exactly x values of p. (Note that $\exists^{=x} p$ binds p but leaves x free.)

Connectives $\phi \rightarrow \psi$ and $\phi \leftrightarrow \psi$ can be expressed as $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$ and $\phi \leftrightarrow \psi \equiv \phi \rightarrow \psi \wedge \psi \rightarrow \phi$. Quantifiers $\exists p.\phi$ and $\forall p.\phi$ can be expressed using $\exists x.(x > 0 \wedge \exists^{=x} p.\phi)$ and $\exists x.(\exists^{=x} p.\top \wedge \exists^{=x} p.\phi)$, respectively.

A sentence is a formula with no free variables. If $w \in \Sigma^*$ and σ is a sentence, we write $w \models \sigma$ if w makes σ true under the intended interpretation.

Definition 9. If σ is a sentence of $\mathsf{FOC}[+;\mathsf{MOD}]$, the language defined by σ is $\{w \mid w \models \sigma\}$.

4.3. Normal form

The part of the logic having to do with position variables is like monadic first-order logic, in which all predicates are monadic (that is, they take only one argument). The part of the logic having to do with count variables is the theory of rational numbers with ordering and addition (but not multiplication). Both of these other logics have useful normal forms: monadic first-order logic has a normal form that uses only one variable (Boolos et al., 2007, p. 274–275), while the theory of rationals with ordering and addition has quantifier elimination (Robinson & Zakon, 1960; Ferrante & Rackoff, 1975). We can combine these two results to get a very simple normal form for $\mathsf{FOC}[+;\mathsf{MOD}]$.

Theorem 1. *Every formula ϕ of $\mathsf{FOC}[+;\mathsf{MOD}]$ is equivalent to a formula of the form*

$$\phi' = \exists x_1 \ldots \exists x_k . \left(\bigwedge_i \exists^{=x_i} p.\psi_i \wedge \chi \right) \tag{5}$$

where

- *Each ψ_i is quantifier-free and has no free count variables.*
- *χ is quantifier-free.*

Proof. See Appendix A. □

It may seem odd that count variables range over rational numbers, when counts are always integers. This technicality simplifies the normal form: if we had used integers, then the part of the logic having to do with count variables would be Presburger arithmetic, and the normal form would require allowing $\mathsf{MOD}^r_m(x)$ on count variables as well.

5. From Transformers to $\mathsf{FOC}[+;\mathsf{MOD}]$

In this section, we prove the following theorem, which sets an upper bound on the expressivity of fixed-precision transformer classifiers.

Theorem 2. *Every language that is recognizable by a fixed-precision transformer classifier is definable by a sentence of $\mathsf{FOC}[+;\mathsf{MOD}]$.*

We don't specify exactly when and how a fixed-precision transformer performs rounding. Our translation to $\mathsf{FOC}[+;\mathsf{MOD}]]$ for the most part can accommodate any rounding scheme, except that in Eq. (6) below, we assume that the averages over positions p are computed exactly, then rounded.

5.1. Representing numbers

Following Merrill & Sabharwal (2022), we use a representation of real numbers with both limited precision and limited range. Limited precision might be justified by the fact that the numbers computed by a neural network are subject to noise (e.g., from randomness in sampling the training data and parameter optimization); limited range is justified by the observation by Hahn (2020) that if all the functions used in a transformer are Lipschitz continuous, then the absolute value of all activations has an upper bound not depending on n. (Appendix D.2 re-proves this result in the presence of layer normalization, which is not in general Lipschitz continuous.)

While Merrill & Sabharwal (2022) use floating point numbers ($p/2$ bits for the mantissa and $p/2$ bits for the exponent, where $p = 16$ or 32), we use a fixed-point representation, with r for the integer part and s for the fractional part. There is no loss of generality, because a floating-point number with a $p/2$-bit mantissa and $p/2$-bit exponent can be converted exactly to a fixed-point number with $p/2 + 2^{p/2}$ bits.

Definition 10. A *fixed-precision number* with r integer bits and s fractional bits is a number in $\mathbb{F}_{r,s} = \{i/2^s \mid -2^{r+s} \leq i < 2^{r+s}\}$. Since r and s are fixed, we normally just write $\mathbb{F}$ in place of $\mathbb{F}_{r,s}$.

We write $\langle x \rangle_i$ for the i-th bit in the two's-complement representation of x. That is,

$$\langle x \rangle_i = \left\lfloor \frac{x}{2^i} \right\rfloor - 2 \left\lfloor \frac{x}{2^{i+1}} \right\rfloor$$

where $\lfloor x \rfloor$ is the greatest integer less than or equal to x.

A neural network, given input $\mathtt{CLS} \cdot w$, computes many real-valued activations, which we can think of as functions $a : \Sigma^* \rightarrow \mathbb{F}$. For each activation a, we will write sentences that test bits of $a(w)$.

Definition 11. If $a : \Sigma^* \rightarrow \mathbb{F}_{r,s}$, we say that a is *defined* by sentences $\langle \sigma^a_k \rangle_{k \in [-s,r]}$ (or just $\langle \sigma^a_k \rangle$ for short) if, for all

$k \in [-s, r]$, $w \models \sigma_k^a$ iff $\langle a(w) \rangle_k = 1$.

Similarly, if $\mathbf{a} \colon \Sigma^n \to \mathbb{F}^{n'}$, we say that $\mathbf{a}$ is defined by $\langle \phi_k^{\mathbf{a}}[p], \omega_k^{\mathbf{a}} \rangle$ if $[\mathbf{a}(w)]_p$ is defined by $\langle \phi_k^{\mathbf{a}}[p] \rangle$ and $[\mathbf{a}(w)]_0$ is defined by $\langle \omega_k^{\mathbf{a}} \rangle$.

The finiteness of $\mathbb{F}$ ensures the following fact, which we use repeatedly:

Proposition 3. *If $a \colon \Sigma^* \to \mathbb{F}$ is defined by $\langle \sigma_k^a \rangle$, then for any function $f \colon \mathbb{F} \to \mathbb{F}$ there are sentences that define $f \circ a$. Similarly, if $b \colon \Sigma^* \to \mathbb{F}$ is defined by $\langle \sigma_k^b \rangle$, and $g \colon \mathbb{F} \times \mathbb{F} \to \mathbb{F}$, there are sentences that define the function $g \circ (a, b)$ (that is, the function that maps w to $g(a(w), b(w))$).*

Proof. Because $\mathbb{F}$ is finite, it is easy but tedious to write sentences that test for all possible inputs and outputs. □

5.2. Input layer

The (function that maps w to the) i-th component of $\mathrm{WE}(w_p)$ is defined by

$$\phi_k[p] = \bigvee_{\substack{a \in \Sigma \\ \langle [\mathrm{WE}(a)]_i \rangle_k = 1}} Q_a(p)$$

and ω_k, which simply encodes the constant value $[\mathrm{WE}(\texttt{CLS})]_i$ in fixed-precision.

Sinusoidal PEs, rounded to the nearest fixed-precision number, can be described using modular predicates. For any i, there exists a period m_i such that for all p, $[\mathrm{PE}(p)]_i = [\mathrm{PE}(p + m_i)]_i$. Then the (function that maps w to the) i-th component of $\mathrm{PE}(p)$ is defined by

$$\phi_k[p] = \bigvee_{\substack{0 \le r < m_i \\ \langle [\mathrm{PE}(r)]_i \rangle_k = 1}} \mathsf{MOD}^r_{m_i}(p)$$

and ω_k, which simply encodes the constant value $[\mathrm{PE}(0)]_i$ in fixed-precision.

5.3. Hidden layers

The position-wise FFNs and residual connections can all be defined using Proposition 3.

In a self-attention layer, the logits (1) can be defined using Proposition 3. The equation for context vectors (2) can be rewritten in terms of averages (not sums, which could overflow):

$$c_q = \frac{\frac{1}{n'} \sum_p \exp s_{qp} W^{(\mathrm{V})} A_{*,p}}{\frac{1}{n'} \sum_p \exp s_{qp}}. \tag{6}$$

Everything in this equation except for averaging ($\frac{1}{n'} \sum_p$) can also be defined using Proposition 3. So the one operation that remains to be defined is averaging over n' positions. Appendix B explains how to do this, one bit at a time. To sum bits across all positions, we use counting quantifiers; to divide by n', we use another kind of number representation whose least-significant digits are n' times smaller than those of fixed-precision numbers.

5.4. Output layer

By composing the constructions from the preceding sections, we obtain formulas $\langle \phi_k[p], \omega_k \rangle$ that define the output of the encoder at all positions. We are only interested in the output at CLS, which is defined by $\langle \omega_k \rangle$, so we can discard the $\phi_k[p]$. By Proposition 3, we can define the top-level sentence, which applies the final sigmoid layer (4) to $\langle \omega_k \rangle$ and tests whether the result is at least $\frac{1}{2}$.

5.5. Complexity analysis

The sentence constructed by Theorem 2 would be quite large if written out in full, because of repeated subformulas. We analyze its size assuming that repeated subformulas can share space.

In the input layer (§5.2), the word embeddings translate to subformulas with total size $O(|\Sigma| d(r + s))$, and the positional encodings, $O(md(r + s))$, where m is the maximum period of any component of PE. In the hidden layers (§5.3), the position-wise FFNs and residual connections translate into subformulas with total size $O(Ld^2F)$, where F is the maximum size of a subformula constructed by Proposition 3, which in the worst case could be exponential in the precision $(r + s)$. The attention layers translate into subformulas with total size $O(LHd^2F)$. Finally, the output layer (§5.4) translates into a subformula of size $O(d^2F)$.

5.6. Relationship to uniform TC^0

We conclude this section by showing that $\mathsf{FOC}[+; \mathsf{MOD}]$ is strictly less expressive than uniform $\mathsf{TC}^0$and therefore a tighter upper bound on fixed-precision transformer encoders than that of Merrill & Sabharwal (2022). (On the other hand, their proof applies to a much more general class of neural networks than ours does.)

(Non-uniform) TC^0 is the class of families of Boolean circuits with majority gates, unlimited fan-in, polynomial size, and constant depth. By *uniform* TC^0 we mean circuit families in TC^0 whose connections can be decided in logarithmic time (Barrington et al., 1990).

Proposition 4. *The language $\{\mathtt{0}^n\mathtt{1}^n \mid n \ge 0\}$ is in uniform TC^0 but not definable in $\mathsf{FOC}[+; \mathsf{MOD}]$.*

Proof. For inclusion in uniform TC^0, we use the fact that uniform TC^0 is equivalent to first-order logic with majority quantifiers, addition, and multiplication, and that majority quantifiers can simulate counting quantifiers (Barrington

et al., 1990, p. 296). Then $\{0^n 1^n\}$ is defined by

$$\exists x.\big(\exists^{=x} p.Q_0(p) \wedge \exists^{=x} p.Q_0(p)\big) \\ \wedge \forall p.\forall q.\big(Q_0(p) \wedge Q_1(q) \rightarrow p < q\big).$$

For non-definability in $\mathsf{FOC}[+;\mathsf{MOD}]$, suppose that $\{0^n 1^n\}$ is definable in $\mathsf{FOC}[+;\mathsf{MOD}]$ by some sentence σ. Let M be the product of all moduli m used in atomic formulas $\mathsf{MOD}_r^m(p)$ used in σ. Then σ cannot distinguish between positions p and $(p+M)$, so it cannot distinguish $w = 0^M 1^M$ and $w' = 1 0^{M-1} 0 1^{M-1}$. Since $w \models \sigma$, it must be the case that $w' \models \sigma$, which is a contradiction. □

6. From $\mathsf{FOC}[+;\mathsf{MOD}]$ to Transformers

In this section, we prove the following theorem, which sets a lower bound on the expressivity of (arbitrary-precision) transformer classifiers.

Theorem 5. *Every language that is definable by a sentence of* $\mathsf{FOC}[+;\mathsf{MOD}]$ *is also recognizable by a transformer classifier.*

By Theorem 1, we can assume

$$\sigma \equiv \exists x_1. \ldots \exists x_k. \left(\bigwedge_i \exists^{=x_i} p.\psi_i[p] \wedge \chi[x_1, \ldots, x_k] \right)$$

where every ψ_i is quantifier-free with one free position variable and no free count variables, and χ is quantifier-free with no free position variables.

Then the proof constructs a transformer classifier with three parts. The first, lowest, part of the network computes the truth values of the $\psi_i[p]$ at every position p. The second part uses uniform self-attention to find each x_i, the number of positions p that make $\psi_i[p]$ true. The third part computes the truth value of χ and of the whole sentence.

We first show how to do this without layer normalization; Appendix D.3 explains how to modify the construction for layer normalization.

6.1. Computing the $\psi_i[p]$

For each $\psi_i[p]$, we construct a transformer encoder that computes its truth-value, in the following sense:

Lemma 6. *For any formula $\psi[p]$ of $\mathsf{FOC}[+;\mathsf{MOD}]$ which is quantifier-free with exactly one free position variable p and no free count variables, there is a transformer encoder T with width d such that, for all $w \in \Sigma^*$ and $p \in [1, |w|]$, $[T(w)]_{d,p} = \mathbb{I}[w \models \psi[p]]$ and $[T(w)]_{d,0} = 0$.*

The proof, given in Appendix C.2, is by induction on subformulas. The cases for $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ invoke the inductive hypothesis for both ψ_1 and ψ_2 and combine them into a single transformer encoder using the following:

Lemma 7. *If T_1 and T_2 are transformer encoders, then there is a transformer encoder, called $T_1 \oplus T_2$, such that*

$$(T_1 \oplus T_2)(w) = \begin{bmatrix} T_1(w) \\ T_2(w) \end{bmatrix}.$$

Proof. See Appendix C.1. □

Then most cases add one hidden layer on top, whose self-attention does nothing ($W^{(\mathrm{V})} = \mathbf{0}$) and whose FFN computes the relevant function.

Thus, for each $\psi_i[p]$, we get an equivalent transformer encoder, which we call Ψ_i.

6.2. Counting quantifiers

The second step is to find the value of each x_i, which is the number of positions p for which $w \models \psi_i[p]$. Construct a transformer encoder U such that, for all p,

$$[U(w)]_{*,p} = \begin{bmatrix} \mathbb{I}\big[w_p = \mathsf{CLS}\big] \\ \mathbf{0}^{k+1} \end{bmatrix}.$$

Use Lemma 7 to form $\Psi = \bigoplus_{i=1}^k \Psi_i \oplus U$, and add one more layer. In the self-attention, $W^{(\mathrm{V})}$ projects and permutes dimensions so that the value vectors are

$$W^{(\mathrm{V})}\,[\Psi(w)]_{*,p} = \begin{bmatrix} \mathbf{0} \\ \mathbb{I}[w \models \psi_1[p]] \\ \vdots \\ \mathbb{I}[w \models \psi_k[p]] \\ \mathbb{I}\big[w_p = \mathsf{CLS}\big] \end{bmatrix}.$$

The self-attention uses uniform attention to average over p ($W^{(\mathrm{Q})} = W^{(\mathrm{K})} = \mathbf{0}$), and the FFN does nothing ($W^{(1)} = b^{(1)} = W^{(2)} = b^{(2)} = \mathbf{0}$). Call this encoder C. Its output, for all p, is

$$[C(w)]_{*,p} = \frac{1}{n'} \begin{bmatrix} \mathbf{0} \\ x_1 \\ \vdots \\ x_k \\ 1 \end{bmatrix}.$$

6.3. Computing χ

The third step is to compute the truth value of χ given the values of the x_i. After the division by n', we can no longer use 1 for true and 0 for false; instead, we use positive numbers for true and negative numbers for false.

Lemma 8. *If χ is a quantifier-free formula of $\mathsf{FOC}[+;\mathsf{MOD}]$ with free count variables $x_1, \ldots, x_k$ and no position variables, then there is a transformer stack X with*

width d such that for any $A \in \mathbb{R}^{d \times n'}$ with

$$A_{*,0} = \alpha \begin{bmatrix} \mathbf{0} \\ x_1 \\ \vdots \\ x_k \\ 1 \end{bmatrix} \qquad x_1, \ldots, x_k \in \mathbb{Z}$$

we have $[X(A)]_{d,0} > 0$ if $\chi[x_1, \ldots, x_k]$ is true, and $[X(A)]_{d,0} < 0$ if $\chi[x_1, \ldots, x_k]$ is false.

Proof. See Appendix C.3. □

Using this lemma, we get a stack of transformer layers equivalent to χ; call it X.

Next, we want to compose C and X. Let d_C and d_X be the width of C and X, respectively, and let $d = \max\{d_C, d_X\}$. If $d_C < d$, construct an encoder Z that outputs $\mathbf{0}^{d-d_C}$ and let $C' = Z \oplus C$ and $X' = X$. Similarly if $d_X < d$. Let $T = X' \circ C'$; then $[T(w)]_{d,0}$ is positive iff the whole sentence is true.

Finally, the output layer Eq. (4) projects $[T(w)]_{*,0}$ to dimension d, so the output probability is greater than $\frac{1}{2}$ iff the whole sentence is true.

6.4. Relationship to counter machines

Bhattamishra et al. (2020) define a kind of counter machine called *simplified stateless counter machine* (SSCM). It has zero or more counters, and upon reading each input symbol a, it increments or decrements each counter by an amount that depends only on a. At the end of the input string, the accept/reject decision depends on whether the counters are zero. Then they prove that any SSCM can be converted to an equivalent transformer. Since our definition of transformers and how they accept strings are slightly different from theirs, we give a slightly different definition of SSCM from theirs. We discuss these differences at the end of this section.

Definition 12. A *simplified stateless k-counter machine* (Merrill, 2020; Bhattamishra et al., 2020), or k-SSCM, is a tuple (Σ, u, F) where

- Σ is a finite alphabet
- $u \colon \Sigma \to \mathbb{Z}^k$ is a counter update function
- $F \subseteq \{0, 1\}^k$ is an acceptance mask.

Definition 13. Let M be a k-SSCM, and let $w = w_1 \cdots w_n$ be an input string. We say that M *accepts* w if there is a sequence $c_0, \ldots, c_n \in \mathbb{Z}^k$ such that

- $c_0 = \mathbf{0}$
- $c_i = c_{i-1} + u(w_i)$ for all $i \in [1, n]$
- $[c_n]_i = 0$ iff $F_i = 0$.

We say that M *recognizes* a language L if $L = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

Proposition 9. *For any simplified stateless k-counter machine (Bhattamishra et al., 2020) there is a sentence of* $\mathsf{FOC}[+; \mathsf{MOD}]$ *that defines the same language.*

Proof. Let $\Sigma = \{a_1, \ldots, a_m\}$, and let $M = (\Sigma, u, F)$ be a k-SSCM. Then the following sentence of $\mathsf{FOC}[+; \mathsf{MOD}]$ is equivalent to M:

$$\sigma = \exists x_1 \ldots . \exists x_m . \left(\bigwedge_{j=1}^{m} \exists^{=x_j} p . Q_{a_j}(p) \wedge \bigwedge_{i=1}^{k} \phi_i \right)$$

$$\phi_i = \begin{cases} u(a_1)x_1 + \cdots + u(a_m)x_m = 0 & F_i = 0 \\ u(a_1)x_1 + \cdots + u(a_m)x_m \neq 0 & F_i = 1. \end{cases}$$

□

On the other hand, $\mathsf{FOC}[+; \mathsf{MOD}]$ can define languages that SSCMs cannot, making it a tighter lower bound than Bhattamishra et al.'s.

Proposition 10. *The language $(\mathtt{01})^*$ is definable in* $\mathsf{FOC}[+; \mathsf{MOD}]$ *but cannot be recognized by any SSCM.*

Proof. The language is definable using the sentence

$$\exists x . (\exists^{=x} p . Q_{\mathtt{0}}(p) \wedge \exists^{=x} p . Q_{\mathtt{1}}(p) \wedge \forall p . \mathsf{MOD}^1_2(p) \leftrightarrow Q_{\mathtt{0}}(p)).$$

To see that this language cannot be recognized by a SSCM, observe that SSCMs are permutation-invariant. That is, let M be a SSCM. For any string $w = w_1 \cdots w_n$ and any permutation π on $[1, n]$, define $\pi(w) = w_{\pi(1)} \cdots w_{\pi(n)}$. Then M accepts w if and only if it accepts $\pi(w)$. Since this means that M cannot distinguish $(\mathtt{01})^n$ from $\mathtt{0}^n\mathtt{1}^n$, it cannot recognize $(\mathtt{01})^*$. □

Bhattamishra et al.'s definition of transformers differs from ours in two ways. First, their transformer encoders have so-called causal (or future) masking, in which each position only attends to the position to the left. Ours do not, and indeed it appears that expressing causal masking would require a binary predicate $p < q$, which would break Theorem 1. We leave investigation of transformer encoders with causal masking for future work. Regardless, masking is not required for simulating SSCMs, as our Proposition 9 holds without it.

Second, whereas our definition of transformer classifier places the output sigmoid layer over `CLS`, theirs places it over the last string position. This is an arbitrary decision,

but as a consequence, their definition of SSCM lets the accept/reject decision depend also on the last input symbol.

Consequently, Theorem 5 does not, strictly speaking, improve on their lower bound. We don't consider this to be a critical issue, however, because it doesn't seem to relate to essential properties of either transformers or counter machines, and it could easily be fixed, for example, by extending the transformer's positional encoding with a flag indicating the end of string and $\mathsf{FOC}[+;\mathsf{MOD}]$ with a predicate indicating the end of string.

7. Related Work

Neural networks have been studied in relation to propositional logic from the start (McCulloch & Pitts, 1943). Much more recently, Barceló et al. (2020) relate graph neural networks to first-order logic with threshold counting quantifiers ($\exists^{\geq k}$ where k is a constant) and at most two variables.

To our knowledge, the only attempts to relate transformers to formal logic are that of Merrill & Sabharwal (2022) and the present paper. But there is a substantial literature on the expressivity of transformers, and in the rest of this section, we review some of this work, limiting our attention to results on transformers as recognizers of formal languages.

7.1. Upper bounds

Transformer encoders under various restrictions have been shown to fall into various language classes. We have already discussed the upper bound of Merrill & Sabharwal (2022) using fixed-precision numbers in §5.6, and review a few others here.

Hao et al. (2022) study transformers with so-called *hard attention*, where each position attends to the position with the highest attention logit. In the case of a tie, the leftmost position wins. They show (generalizing a result by Hahn (2020) on PARITY and the Dyck language with two pairs of brackets) that such transformers recognize languages in non-uniform AC^0 (that is, families of Boolean circuits with unlimited fan-in, polynomial size, and constant depth).

Merrill et al. (2022) study transformers with *saturated attention* where, in the case of a tie, attention is distributed evenly among the tied positions. Additionally, they assume that all activations are numbers of the form $x/2^y$ where x and y are integers, with certain operations (reciprocal, square root) rounded to the nearest multiple of $1/2^y$. They show that such transformers recognize languages in non-uniform TC^0. In subsequent work (Merrill & Sabharwal, 2023), they show that transformers using full (softmax) attention and numbers with $O(\log n)$ bits recognize languages in logspace-uniform TC^0.

Also worth mentioning is a result by Hahn (2020); he considers transformer classifiers whose activation functions are Lipschitz continuous (that is, if layer normalization is used, then $\epsilon > 0$) and which always accept or reject strings with some (arbitrarily small, but fixed) margin. He shows that such transformer classifiers cannot recognize PARITY or the Dyck language with two pairs of brackets.

All of the above upper bounds require some modification to the definition of transformer. Ours (Theorem 2) is no exception: although we use full (softmax) attention, we limit numbers to fixed-precision. Relaxing this restriction would break Proposition 3 and is left for future investigation.

7.2. Lower bounds

We have discussed the lower bound of Bhattamishra et al. (2020) already in §6.4. The other lower bounds that we are aware of involve extensions of transformers. First, Chiang & Cholak (2022) showed that transformers whose PEs include a p/n' component can recognize PARITY.

Second, RASP (Weiss et al., 2021) is a programming language that can be compiled to transformers with saturated attention and several other extensions that appear to increase their expressivity. Their attention weights are directly computed from the previous layer, and are not restricted to be dot-products of query and key vectors; this allows compilation of expressions involving binary predicates like $p = q$ or $p < q$. Their position-wise FFNs are allowed to compute arbitrary functions, the rationale being that they can be approximated by ReLU FFNs by the universal approximation theorem.

In contrast, our lower bound here follows the definition of a transformer encoder fairly strictly; the only departure is allowing the PE to have sine/cosine waves with different frequencies than the original definition.

Third, Pérez et al. (2021) consider transformers with saturated attention and several extensions. But more importantly, their result concerns, not encoders, but encoder–decoders. The encoder reads a string w, and the decoder is allowed to run for an arbitrary number of steps before making an accept/reject decision. This makes the model much more powerful: Pérez et al. (2021) show that it can simulate a Turing machine.

The use of a decoder is of course standard and not an "extension." But our result and theirs are in no way contradictory; they simply concern two very different configurations of transformers. Intuitively, one could liken a transformer encoder to a system that is prompted with a string and must immediately accept or reject, whereas a transformer encoder–decoder could be likened to a system that can "think step by step" (Kojima et al., 2022) before generating a final answer.

8. Discussion

8.1. Relationship with other complexity classes

We have already discussed the relationship of $\mathsf{FOC}[+;\mathsf{MOD}]$ with TC^0 (§5.6) and SSCMs (§6.4). We can further relate $\mathsf{FOC}[+;\mathsf{MOD}]$ to the classes of regular languages and uniform AC^0:

Proposition 11. *The class of languages recognizable by* $\mathsf{FOC}[+;\mathsf{MOD}]$ *is not comparable with the class of regular languages.*

Proof. The language MAJORITY, containing those strings with more 1's than 0's, is definable in $\mathsf{FOC}[+;\mathsf{MOD}]$ by the sentence $\exists x.\exists y.\,(\exists^{=x} p.Q_0(p) \wedge \exists^{=y} p.Q_1(p) \wedge x > y)$, but is not regular. On the other hand, 0^*1^* is regular but not definable in $\mathsf{FOC}[+;\mathsf{MOD}]$, by an argument similar to Proposition 4. □

Proposition 12. *The class of languages recognizable by* $\mathsf{FOC}[+;\mathsf{MOD}]$ *is not comparable with uniform* AC^0.

Proof. The languages used in the proof of Proposition 11 apply here as well: MAJORITY is not in AC^0 (Furst et al., 1984), but 0^*1^* is in uniform AC^0, because AC^0 is equivalent to FO+BIT, which includes the sentence $\exists p.\forall q.(q < p \rightarrow Q_0(p) \wedge q \geq p \rightarrow Q_1(p))$. □

8.2. Transformer variants

Section 7.1 mentioned several restrictions of transformers, proposed to make finding upper bounds easier. We think these are interesting in their own right, and it would be worthwhile to clarify the relationships among them. One relationship is already implied by Theorems 2 and 5. The translation from fixed-precision transformers to $\mathsf{FOC}[+;\mathsf{MOD}]$ to arbitrary-precision transformers produces networks that only use uniform attention, which is a special case of saturated attention. So fixed-precision transformers are at most as powerful as saturated-attention transformers.

Similarly, §7.2 mentioned several extensions of transformers, and curiously, all three of these previous lower bounds include p or p/n' in their PEs (also cf. Yun et al., 2020). Intuitively, this gives them the ability to translate between counts and positions, and we think this extension merits further study, both theoretical and experimental.

8.3. Next steps

Nonetheless, our ultimate goal is to exactly characterize the expressivity of unrestricted transformers with rational weights. It might be thought that rational weights would add too much power, since they can store an unbounded amount of information; for example, an RNN with rational weights is equivalent to a Turing machine (Siegelmann & Sontag, 1995). But this is true only if it is allowed to run for arbitrarily many time steps; if it runs for n time steps, it is intermediate in power between real-time Turing machines and real-time RAM machines (Chen et al., 2017). Since a transformer encoder only has fixed depth L, we think it is reasonable to hope for a logic that is both equivalent to rational-weighted transformers and has useful and interesting properties.

$\mathsf{FOC}[+;\mathsf{MOD}]$ is a significant step in that direction. By Theorem 2, we know that it can at least express anything that real-world transformer encoders can, and by Theorem 5, we know that it does not have any excess expressivity that does anything "un-transformer-like." Crucial to these results is the normal form of Theorem 1. Just as (with a fixed number of inputs) arbitrary Boolean functions can be expressed as a disjunction of conjunctions, or arbitrary continuous functions can be approximated by a FFN, in a similar way (with variable-length input strings), our normal form allows some fairly complicated properties to be expressed in a very simple form that can be mapped to transformers. We speculate that a normal form result along the lines of Theorem 1 will make an exact characterization of unrestricted transformer encoders possible, and that it will provide insight into how still more complex properties of strings can be computed by transformers.

Acknowledgements

We thank Brian DuSell and the anonymous reviewers for their comments. Peter Cholak was partially supported by NSF grant DMS-1854136, and Anand Pillay was supported by NSF grants DMS-1760212 and DMS-2054271.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. In *NIPS 2016 Deep Learning Symposium*, 2016. URL https://arxiv.org/abs/1607.06450.

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J., and Silva, J.-P. The logical expressiveness of graph neural networks. In *Proc. ICLR*, 2020. URL https://openreview.net/pdf?id=r1lZ7AEKvB.

Barrington, D. A. M., Immerman, N., and Straubing, H. On uniformity within NC^1. *J. Computer and System Sciences*, 41(3), 1990. doi: 10.1016/0022-0000(90)90022-D.

Bhattamishra, S., Ahuja, K., and Goyal, N. On the ability and limitations of Transformers to recognize formal languages. In *Proc. EMNLP*, 2020. doi: 10.18653/v1/2020.emnlp-main.576.

Boolos, G. S., Burgess, J. P., and Jeffrey, R. C. *Computability and Logic*. Cambridge Univ. Press, 5th edition, 2007.

Büchi, J. R. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6, 1960. doi: 10.1002/malq.19600060105.

Chen, Y., Gilroy, S., Knight, K., and May, J. Recurrent neural networks as weighted language recognizers, 2017. URL https://arxiv.org/abs/1711.05408v1. arXiv:1711.05408v1. Earlier version of a paper presented at NAACL HLT 2018.

Chiang, D. and Cholak, P. Overcoming a theoretical limitation of self-attention. In *Proc. ACL*, 2022. doi: 10.18653/v1/2022.acl-long.527.

Ferrante, J. and Rackoff, C. A decision procedure for the first order theory of real addition with order. *SIAM J. Computing*, 4(1), March 1975. doi: 10.1137/0204006.

Forcada, M. L. and Carrasco, R. C. Finite-state computation in analog neural networks: Steps towards biologically plausible models? In Wermter, S., Austin, J., and Willshaw, D. (eds.), *Emergent Neural Computational Architectures Based on Neuroscience: Towards Neuroscience-Inspired Computing*. Springer, 2001. doi: 10.1007/3-540-44597-8_34.

Furst, M., Saxe, J. B., and Sipser, M. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984. doi: 10.1007/BF01744431.

Hahn, M. Theoretical limitations of self-attention in neural sequence models. *Trans. ACL*, 8, 2020. doi: 10.1162/tacl_a_00306.

Hao, Y., Angluin, D., and Frank, R. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. ACL*, 10, 2022. doi: 10.1162/tacl_a_00490.

Immerman, N. *Descriptive Complexity*. Springer, 1999.

Kleene, S. C. Representation of events in nerve nets and finite automata. In Shannon, C. E. and McCarthy, J. (eds.), *Automata Studies*, number 34 in Annals of Mathematics Studies. Princeton Univ. Press, 1956. doi: 10.1515/9781400882618-002.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. In *Proc. NeurIPS*, pp. 22199–22213, 2022. URL https://arxiv.org/pdf/2207.00729.pdf.

McCulloch, W. and Pitts, W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 1943. doi: 10.1007/BF02478259.

Merrill, W. On the linguistic capacity of real-time counter automata, 2020. URL https://arxiv.org/abs/2004.06866. arXiv:2004.06866.

Merrill, W. and Sabharwal, A. Transformers can be translated to first-order logic with majority quantifiers, 2022. URL https://arxiv.org/abs/2210.02671v3. arXiv:2210.02671v3.

Merrill, W. and Sabharwal, A. The parallelism tradeoff: Limitations of log-precision transformers. *Trans. ACL*, 2023. URL https://arxiv.org/pdf/2207.00729.pdf. To appear.

Merrill, W., Sabharwal, A., and Smith, N. A. Saturated transformers are constant-depth threshold circuits. *Trans. ACL*, 10, 2022. doi: 10.1162/tacl_a_00493.

Nguyen, T. Q. and Salazar, J. Transformers without tears: Improving the normalization of self-attention. In *Proc. International Workshop on Spoken Language Translation (IWSLT)*, 2019. doi: 10.5281/zenodo.3525484.

Pérez, J., Barceló, P., and Marinkovic, J. Attention is Turing-complete. *J. Machine Learning Research*, 22 (75), 2021. URL https://jmlr.org/papers/v22/20-302.html.

Robinson, A. and Zakon, E. Elementary properties of ordered abelian groups. *Trans. AMS*, 96, 1960. doi: 10.1090/S0002-9947-1960-0114855-0.

Schwartz, R., Thomson, S., and Smith, N. A. Bridging CNNs, RNNs, and weighted finite-state machines. In *Proc. ACL*, 2018. doi: 10.18653/v1/P18-1028.

Siegelmann, H. and Sontag, E. On the computational power of neural nets. *J. Computer and System Sciences*, 50(1), 1995. doi: 10.1006/jcss.1995.1013.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Proc. NeurIPS*, 2017. URL https://papers.nips.cc/paper/7181-attention-is-all-you-need.

Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., and Chao, L. S. Learning deep Transformer models for machine translation. In *Proc. ACL*, 2019. doi: 10.18653/v1/P19-1176.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking like Transformers. In *Proc. ICML*, 2021. URL https://proceedings.mlr.press/v139/weiss21a.html.

Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S. J., and Kumar, S. Are Transformers universal approximators of sequence-to-sequence functions? In *Proc. ICLR*, 2020. URL https://openreview.net/pdf?id=ByxRM0Ntvr.

A. Proof of Theorem 1

Theorem 1 says that every formula ϕ of $\mathsf{FOC}[+;\mathsf{MOD}]$ is equivalent to a formula of the form

$$\phi' = \exists x_1. \ldots \exists x_k. \left(\bigwedge_i \exists^{=x_i} p. \psi_i \wedge \chi \right) \tag{7}$$

where

- Each ψ_i is quantifier-free and has no free count variables.
- χ is quantifier-free.

The construction of ϕ' proceeds in three steps. First, move counting quantifiers inward until they are of the form $\exists^{=x_i} p. \psi_i$ where ψ_i is quantifier-free and has no free count variables. Second, move all subformulas $\exists^{=x_i} p. \psi_i$ outward, leaving behind χ, as in Eq. (7). Third, eliminate quantifiers from χ.

The first step is analogous to the proof that a formula of monadic first-order logic can be converted to one with only one variable (Boolos et al., 2007, p. 274–275), which moves existential quantifiers inwards using the two following facts: $\exists x. (\phi \vee \psi) \equiv (\exists x. \phi) \vee (\exists x. \psi)$, and if x does not occur free in ϕ, then $\exists x. (\phi \wedge \psi) \equiv \phi \wedge (\exists x. \psi)$. With counting quantifiers, neither of the above holds, but the following two lemmas serve similar purposes.

Lemma 13. *Any formula $\phi = \exists^{=x} p. \bigvee_{i=1}^{l} \phi_i$ is equivalent to a formula*

$$\phi' = \exists x_1. \ldots \exists x_m. \left(\bigwedge_{j=1}^{m} \exists^{=x_j} p. \psi_j \wedge \chi \right) \tag{8}$$

where the x_i are fresh count variables, each ψ_j is a conjunction of a subset of the ϕ_i, and χ has no free position variables.

Proof. By induction on l. The case $l = 1$ is trivial. For $l > 1$, write

$$\begin{aligned}
\phi &= \exists^{=x} p. \left(\bigvee_{i=1}^{l-1} \phi_i \vee \phi_l \right) \\
&\equiv \exists x_1. \exists x_2. \exists x_3. \left(\exists^{=x_1} p. \bigvee_{i=1}^{l-1} \phi_i \wedge \exists^{=x_2} p. \phi_l \wedge \exists^{=x_3} p. \left(\bigvee_{i=1}^{l-1} \phi_i \wedge \phi_l \right) \wedge x = x_1 + x_2 - x_3 \right) \\
&\equiv \exists x_1. \exists x_2. \exists x_3. \Bigg(\underbrace{\exists^{=x_1} p. \bigvee_{i=1}^{l-1} \phi_i}_{*} \wedge \exists^{=x_2} p. \phi_l \wedge \underbrace{\exists^{=x_3} p. \bigvee_{i=1}^{l-1} (\phi_i \wedge \phi_l)}_{*} \wedge x = x_1 + x_2 - x_3 \Bigg)
\end{aligned}$$

and use the inductive hypothesis on the subformulas marked $*$ to put this into the form (8). □

Lemma 14. *If p does not occur free in ϕ, then*

$$\exists^{=x} p. (\phi \wedge \psi) \equiv (\neg\phi \wedge x = 0) \vee (\phi \wedge \exists^{=x} p. \psi).$$

Using these facts, we can prove the following:

Proposition 15. *Every formula is equivalent to a formula in which*

- *in every subformula $\exists x. \phi$, there are no free position variables.*
- *in every subformula $\exists^{=x} p. \phi$, ϕ is quantifier-free and the only free variable in ϕ is p itself.*

Proof. By induction on subformulas. Call a subformula *clear* if it has the two properties listed above.

Case $\exists x.\phi$: Write ϕ as a Boolean combination of subformulas $\phi_1, \ldots, \phi_m$ that are either atomic or start with quantifiers. We can put ϕ into DNF in terms of the ϕ_i and distribute the $\exists x$ over the disjuncts. By the induction hypothesis, each of the ϕ_i is equivalent to a ϕ_i' which is clear. So $\phi \equiv \bigvee_{j=1}^{m} \exists x.\ \bigwedge_k \psi_{jk}$ where each ψ_{jk} is one of the ϕ_i'. Consider each ψ_{jk}.

- If ψ_{jk} is $P(p)$ or $\neg P(p)$, then it can be moved out of the $\exists x$.
- If ψ_{jk} starts with $\exists y$, $\neg\exists y$, $\exists^{=y} p$, or $\neg\exists^{=y} p$, then (because ψ_{jk} is clear) it has no free position variables and does not need to be moved.
- If ψ_{jk} is one of $y = z$, $y < z$, $y_1 + y_2 = z$, or their negations, then it has no free position variables and does not need to be moved.

Thus we have constructed a subformula equivalent to $\exists x.\phi$ that is clear.

Case $\exists^{=x} p.\phi$: Again, write ϕ as a Boolean combination of subformulas ϕ_i that are either atomic or start with quantifiers, put ϕ into DNF in terms of the ϕ_k. By the induction hypothesis, each of the ϕ_i is equivalent to a ϕ_i' which is clear. Use Lemma 13 to obtain

$$\phi \equiv \exists x_1. \cdots \exists x_m. \left(\bigwedge_j \exists^{=x_j} p. \bigwedge_k \psi_{jk} \wedge \chi \right)$$

where each ψ_{jk} is one of the ϕ_i'. Consider each ψ_{jk}.

- If ψ_{jk} starts with $\exists y$, $\neg\exists y$, $\exists^{=y} p$, or $\neg\exists^{=y} p$, then it has no free position variables and can be moved out of the $\exists^{=x_j} p$ using Lemma 14.
- If ψ_{jk} is one of $y = z$, $y < z$, $y_1 + y_2 = z$, or their negations, then it can be moved out of the $\exists^{=x_j} p$ using Lemma 14.
- If ϕ_{jk} is $P(q)$ or $\neg P(q)$ where $q \neq p$, then it can be moved out of the $\exists^{=x} p$ using Lemma 14, and also out of the $\exists x_i$.
- If ϕ_{jk} is $P(p)$ or $\neg P(p)$, then it only has free variable p and does not need to be moved.

Thus we have constructed a subformula equivalent to $\exists^{=x} p.\phi$ that is clear. □

This completes the first step. For the second step, let χ be the formula obtained as follows: for every subformula $\exists^{=x_i} p.\psi_i$, let x_i' be a fresh count variable and replace the subformula with $x_i = x_i'$. Thus we have

$$\phi \equiv \exists x_1'. \ldots \exists x_k'. \left(\bigwedge_i \exists^{=x_i} p.\psi_i \wedge \chi \right)$$

which is almost in the desired form except that χ still has quantifiers.

The third step is the following:

Theorem 16 (Ferrante & Rackoff, 1975)**.** *For any formula χ with no position variables (free or bound), there is a quantifier-free formula χ' equivalent to χ.*

Apply this procedure to χ and call the result χ'. Finally, let

$$\phi' = \exists x_1'. \ldots \exists x_k'. \left(\bigwedge_i \exists^{=x_i'} p.\psi_i \wedge \chi' \right).$$

B. Expressing averages in FOC[+; MOD]

To compute averages across n positions, we define a new kind of numeric representation, called *extra-precision numbers*, that have an *extra digit*, ranging from 0 to n with $\frac{1}{n'}$ of the place value of the least-significant bit of a fixed-precision number.

Definition 14. An *extra-precision number* with r integer bits, s fractional bits, and extra digit with base n' is a number in $\mathbb{E}_{r,s,n'} = \left\{ \frac{i}{2^m n'} \;\middle|\; -2^{\ell+m} n' \le i < 2^{\ell+m} n' \right\}$. For any $a \in \mathbb{E}_{r,s,n'}$, we define

$$\langle a \rangle_{\text{extra}} = \lfloor a \cdot 2^s \cdot n' \rfloor - \lfloor a \cdot 2^s \rfloor \cdot n'$$
$$\langle a \rangle_{\text{fixed}} = \lfloor a \cdot 2^s \rfloor \cdot 2^{-s}.$$

For example, in $\mathbb{E}_{1,2,3}$, the extra digit has place value $\frac{1}{2^{2\cdot 3}} = \frac{1}{12}$. The number $\frac{17}{12}$ belongs to $\mathbb{E}_{1,2,3}$ and can be represented as 01.012, because $\frac{17}{12} = 1\cdot 1 + 0\cdot\frac{1}{2} + 1\cdot\frac{1}{4} + 2\cdot\frac{1}{12}$. Then $\left\langle \frac{17}{12} \right\rangle_{\text{extra}} = 2$ and $\left\langle \frac{17}{12} \right\rangle_{\text{fixed}} = \frac{5}{4}$ (or 01.01 in binary). Its negation, $-\frac{17}{12}$, can be represented as 10.101. The sign bit can be thought of as having place value -2, and $-\frac{17}{12} = -1\cdot 2 + 0\cdot 1 + 1\cdot\frac{1}{2} + 0\cdot\frac{1}{4} + 1\cdot\frac{1}{12}$.

Definition 15. If $a\colon \Sigma^* \to \mathbb{E}_{r,s,n'}$, we say that a is *defined* by sentences σ_k^a for $k \in [-s, r]$ and $\phi^a_{\text{extra}}[x]$ (or $\langle \sigma_k^a, \phi^a_{\text{extra}}[x] \rangle$ for short) if $\langle \sigma_k^a \rangle$ defines $\langle a(w) \rangle_{\text{fixed}}$, and $w \models \phi^a_{\text{extra}}[x]$ iff $x = \langle a(w) \rangle_{\text{extra}}$.

Adding or subtracting two extra-precision numbers is possible because $\mathsf{FOC}[+;\mathsf{MOD}]$ has addition of count variables:

Proposition 17. *Let $a\colon \Sigma^* \to \mathbb{E}_{r,s,n'}$ be defined by $\langle \sigma_k^a, \phi^a_{extra}[x] \rangle$ and $b\colon \Sigma^* \to \mathbb{E}_{r,s,n'}$ be defined by $\langle \sigma_k^b, \phi^b_{extra}[x] \rangle$. If, for all w, $(a+b)(w) \in \mathbb{E}_{r,s,n'}$, then $(a+b)$ is defined by some $\langle \sigma_k^{a+b}, \phi^{a+b}_{extra} \rangle$, and similarly for $(a-b)$.*

Proof. First, the extra digit of $(a+b)(w)$ is obtained by adding the extra digits of $a(w)$ and $b(w)$; if the sum is n' or more, then subtract n' and carry a 1 to the least-significant bit. Thus, let

$$\phi^{a+b}_{\text{extra}}[z] = \exists x.\exists y.\exists n.\Big(\sigma^a_{\text{extra}}[x] \wedge \sigma^b_{\text{extra}}[y] \wedge \exists^{=n} p.\top \wedge \Big(\underbrace{(x+y < n+1 \wedge z = x+y)}_{\text{no carry}} \vee \underbrace{(x+y \ge n+1 \wedge z = x+y-(n+1))}_{\text{carry}}\Big)\Big).$$

As for the remaining bits, by Proposition 3, $\langle a \rangle_{\text{fixed}} + \langle b \rangle_{\text{fixed}}$ is definable by some $\langle \sigma_k^0 \rangle$. Similarly, if there was a carry from the extra digit, $\langle a \rangle_{\text{fixed}} + \langle b \rangle_{\text{fixed}} + 1$ is definable by some $\langle \sigma_k^1 \rangle$. Then let

$$\sigma_k^{a+b} = \exists x.\exists y.\exists n.\Big(\sigma^a_{\text{extra}}[x] \wedge \sigma^b_{\text{extra}}[y] \wedge \exists^{=n} p.\top \wedge \Big(\underbrace{(x+y < n+1 \wedge \sigma_k^0)}_{\text{no carry}} \vee \underbrace{(x+y \ge n+1 \wedge \sigma_k^1)}_{\text{carry}}\Big)\Big).$$

To get subtraction, it suffices to define negation. Recall that in two's complement representation, negation means inverting all the digits and adding 1. For the extra digit, "inverting" means subtracting from n. If the extra digit is 0, invert it to get n, then increment it to 0 with a carry to the least-significant bit.

$$\phi^{-b}_{\text{extra}}[z] = \exists y.\exists n.\Big(\sigma^b_{\text{extra}}[y] \wedge \exists^{=n} p.\top \wedge \Big(\underbrace{(y > 0 \wedge z = n+1-y)}_{\text{no carry}} \vee \underbrace{(y = 0 \wedge z = 0)}_{\text{carry}}\Big)\Big).$$

Write $\sim$ for bitwise negation. By Proposition 3, $\sim \langle b \rangle_{\text{fixed}}$ is definable by some $\langle \sigma_k^0 \rangle$, and $\sim \langle b \rangle_{\text{fixed}} + 1$ is definable by some $\langle \sigma_k^1 \rangle$. Then

$$\sigma_k^{-b} = \exists y.\exists n.\left(\sigma^b_{\text{extra}}[y] \wedge \exists^{=n} p.\top \wedge \Big(\underbrace{(y > 0 \wedge \sigma_k^0)}_{\text{no carry}} \vee \underbrace{(y = 0 \wedge \sigma_k^1)}_{\text{carry}}\Big)\right). \qquad \square$$

Finally, we can show how to define averages over n' positions.

Proposition 18. *Given a function $\mathbf{a}\colon \Sigma^n \to \mathbb{F}^{n'}_{r,s}$, defined by $\langle \phi_k^{\mathbf{a}}[p], \omega_k^{\mathbf{a}} \rangle$, the function*

$$\bar{\mathbf{a}}\colon \Sigma^* \to \mathbb{E}_{r,s,n'}$$
$$w \mapsto \frac{1}{n'} \sum_{i=1}^{n'} [\mathbf{a}(w)]_p$$

is approximated with error at most 2^{-s} by a function that is definable by some $\langle \sigma_k^{\bar{\mathbf{a}}} \rangle$.

Proof. Observe that

$$[\mathbf{a}(w)]_p = -\left\langle [\mathbf{a}(w)]_p \right\rangle_r \cdot 2^r + \sum_{k=-s}^{r-1} \left\langle [\mathbf{a}(w)]_p \right\rangle_k \cdot 2^k$$

so the average can be written as

$$\begin{aligned}
\bar{\mathbf{a}}(w) &= \frac{1}{n'} \sum_{p=0}^{n'} [\mathbf{a}(w)]_p \\
&= -\frac{1}{n'} \sum_{p=0}^{n'} \left\langle [\mathbf{a}(w)]_p \right\rangle_r \cdot 2^r + \sum_{k=-s}^{r-1} \frac{1}{n'} \sum_{p=0}^{n'} \left\langle [\mathbf{a}(w)]_p \right\rangle_k \cdot 2^k \\
&= \underbrace{-\frac{1}{2^s n'} \sum_{p=0}^{n'} \left\langle [\mathbf{a}(w)]_p \right\rangle_r}_{v_r} \cdot 2^{r+s} + \sum_{k=-s}^{r-1} \underbrace{\frac{1}{2^s n'} \sum_{p=0}^{n'} \left\langle [\mathbf{a}(w)]_p \right\rangle_k}_{v_k} \cdot 2^{k+s}. \qquad (9)
\end{aligned}$$

Each v_k is the sum of all the k-th bits, written in the extra digit's place. It is defined by

$$\begin{aligned}
\sigma_k^{v_k} &= \bot \\
\phi_{\text{extra}}^{v_k}[x] &= \exists y. \left((\omega_k^{\mathbf{a}} \wedge x = y + 1 \vee \neg \omega_k^{\mathbf{a}} \wedge x = y) \wedge \exists^{=y} p. \phi_k^{\mathbf{a}}[p] \right).
\end{aligned}$$

Then use Proposition 17 to define Eq. (9). The multiplications by powers of 2 can be accomplished by repeated addition, as can the summation over k. □

C. Proofs for Section 6 (From $\mathsf{FOC}[+;\mathsf{MOD}]$ to Transformers)

C.1. Proof of Lemma 7

If one encoder is less deep than the other, add layers to it that compute the identity function: For the self-attention, just set $W^{(\mathrm{V})} = \mathbf{0}$, and for the FFN, set $W^{(1)} = \mathbf{0}$ and $b^{(1)} = \mathbf{0}$. In both cases, the sublayer computes the identity function thanks to the residual connections.

Concatenate the word and position vectors:

$$\mathrm{WE}(a) = \begin{bmatrix} \mathrm{WE}_1(a) \\ \mathrm{WE}_2(a) \end{bmatrix} \qquad \mathrm{PE}(p) = \begin{bmatrix} \mathrm{PE}_1(p) \\ \mathrm{PE}_2(p) \end{bmatrix}.$$

Although we only ever concatenate layers whose self-attentions compute the identity function, we show how to concatenate self-attentions for completeness. For each pair of multi-head self-attentions, $\mathrm{SA}_1^{(1)}, \ldots, \mathrm{SA}_1^{(H_1)}$ and $\mathrm{SA}_2^{(1)}, \ldots, \mathrm{SA}_2^{(H_2)}$, create a multi-head self-attention with $H_1 + H_2$ heads:

$$W^{(h,\mathrm{Q})} = \begin{bmatrix} W_1^{(h,\mathrm{Q})} & \mathbf{0} \end{bmatrix} \quad (1 \le h \le H_1) \qquad W^{(h,\mathrm{Q})} = \begin{bmatrix} \mathbf{0} & W_2^{(h-H_1,\mathrm{Q})} \end{bmatrix} \quad (H_1 + 1 \le h \le H_1 + H_2)$$

$$W^{(h,\mathrm{K})} = \begin{bmatrix} W_1^{(h,\mathrm{K})} & \mathbf{0} \end{bmatrix} \qquad W^{(h,\mathrm{K})} = \begin{bmatrix} \mathbf{0} & W_2^{(h-H_1,\mathrm{K})} \end{bmatrix}$$

$$W^{(h,\mathrm{V})} = \begin{bmatrix} W_1^{(h,\mathrm{V})} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \qquad W^{(h,\mathrm{V})} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & W_2^{(h-H_1,\mathrm{V})} \end{bmatrix}$$

For each FFN_1 and FFN_2, create a FFN:

$$W^{(1)} = \begin{bmatrix} W_1^{(1)} & \mathbf{0} \\ \mathbf{0} & W_2^{(1)} \end{bmatrix} \qquad b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} W_1^{(2)} & \mathbf{0} \\ \mathbf{0} & W_2^{(2)} \end{bmatrix} \qquad b^{(2)} = \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \end{bmatrix}.$$

C.2. Proof of Lemma 6

The following lemma implies that we can always modify a FFN to cancel out its residual connection:

Lemma 19. *If $f : \mathbb{R}^d \to \mathbb{R}^d$ is a FFN, there is a FFN f_- such that $f_-(x) = f(x) - x$.*

Proof.

$$W_-^{(1)} = \begin{bmatrix} W^{(1)} \\ \mathbf{I} \\ -\mathbf{I} \end{bmatrix} \qquad b_-^{(1)} = \begin{bmatrix} b^{(1)} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$
$$W_-^{(2)} = \begin{bmatrix} W^{(2)} & -\mathbf{I} & \mathbf{I} \end{bmatrix} \qquad b_-^{(2)} = b^{(2)}.$$

□

To prove Lemma 6, we first prove the following slightly modified claim, by induction on subformulas:

Lemma 20. *For any formula $\psi[p]$ of $\mathsf{FOC}[+;\mathsf{MOD}]$ which is quantifier-free with exactly one free position variable p and no free count variables, there is a transformer encoder T with width d such that, for all $w \in \Sigma^*$ and $p \in [1, |w|]$, $[T(w)]_{d,p} = \mathbb{I}[w \models \psi[p]]$.*

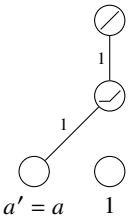
Then the final step will be to set $[T(w)]_{d,0} = 0$.

In the following cases, we give diagrams of FFNs rather than writing out their weight matrices. In these diagrams, a node ◯ with a value underneath it stands for the component of the input vector that contains that value. A node ◯ is a ReLU unit, and a node ◯ is a linear unit. Edges are connections, with their connection weights written next to them. If a unit has nonzero bias, the bias is written next to it. When the residual connection is not cancelled out using Lemma 19, we draw the residual connection as an edge with weight 1.

Case $\psi = Q_a(p)$: Construct a transformer encoder with just an input layer ($L = 0$):

$$\mathrm{WE}(a') = \begin{bmatrix} \mathbb{I}[a' = a] \\ 0 \end{bmatrix}$$
$$\mathrm{PE}(p) = \begin{bmatrix} \sin 0 \\ \cos 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Then add one hidden layer, whose self-attention does nothing ($W^{(\mathrm{V})} = \mathbf{0}$) and whose FFN is

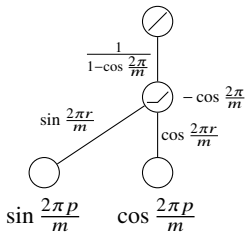


Apply Lemma 19 to cancel out the residual connection.

Case $\psi = \mathsf{MOD}_m^r(p)$: Any function on integer positions with period m can be expressed as a linear function of a sinusoidal PE with width $2m$ using a discrete Fourier series. But here we exploit ReLUs to give a more compact network. Let

$$\mathrm{WE}(a) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\mathrm{PE}(p) = \begin{bmatrix} \sin \frac{2\pi p}{m} \\ \cos \frac{2\pi p}{m} \end{bmatrix}.$$

Then add one hidden layer, whose self-attention does nothing, and whose FFN is



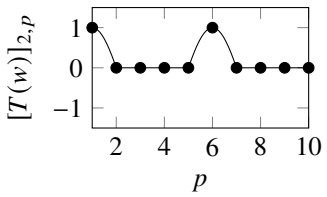
Apply Lemma 19 to cancel out the residual connection. So (using the identity $\cos(x-y) = \cos x \cos y + \sin x \sin y$) the output vectors are

$$[T(w)]_{*,p} = \begin{bmatrix} 0 \\ \frac{\max\left\{0, \cos\frac{2\pi(p-r)}{m} - \cos\frac{2\pi}{m}\right\}}{1-\cos\frac{2\pi}{m}} \end{bmatrix}$$

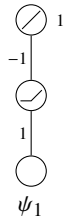
and for p an integer, this simplifies to

$$[T(w)]_{*,p} = \begin{bmatrix} 0 \\ \mathbb{I}[p \equiv_m r] \end{bmatrix}.$$

For example, the graph below shows the case $r = 1$, $m = 5$:

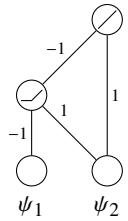


Case $\psi = \neg\psi_1$: By the induction hypothesis, let T_1 be a transformer encoder that computes ψ_1. Add one new layer. The self-attention does nothing ($W^{(V)} = \mathbf{0}$), and the FFN performs the negation:

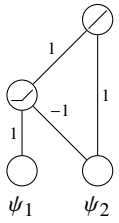


Then apply Lemma 19 to cancel out the residual connection. This computes the negation in the last dimension.

Case $\psi = \psi_1 \wedge \psi_2$: By the induction hypothesis, let T_1 and T_2 be transformer encoders that compute ψ_1 and ψ_2 (respectively). By Lemma 7, we construct $T_1 \oplus T_2$. Then we add one new layer. The self-attention does nothing ($W^{(V)} = \mathbf{0}$), and the FFN computes the minimum of its two inputs. In this case, we do not use Lemma 19, as we make use of the residual connection.



Case $\psi = \psi_1 \vee \psi_2$: The FFN computes the maximum of its two inputs:

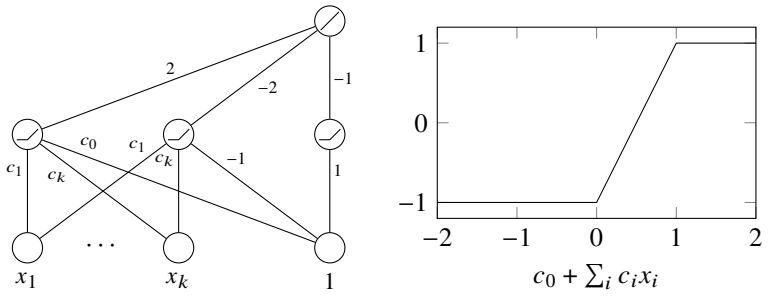


Finally, to set $[T(w)]_{d,0} = 0$, construct a transformer encoder N such that, for all p, $[N(w)]_{*,p} = \left[\mathbb{I}\left[w_p \neq \texttt{CLS}\right]\right]$, and use Lemma 7 to form $T \oplus N$. Add a new layer whose self-attention does nothing, and whose FFN computes the minimum, just as in the case for $\wedge$ above.

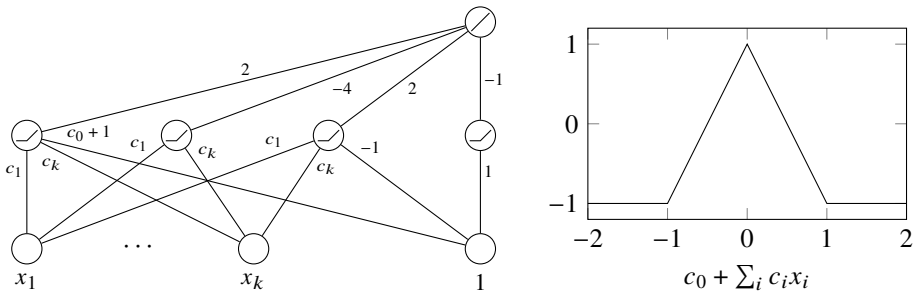
C.3. Proof of Lemma 8

By induction on the structure of χ. For simplicity, we set $\alpha = 1$; since all the FFNs do not have bias, the construction will work for any $\alpha > 0$ as well.

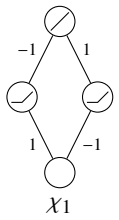
Case $c_0 + \sum_i c_i x_i > 0$: Add a new layer whose self-attention does nothing and whose FFN (after applying Lemma 19) computes the piecewise linear function shown at right.



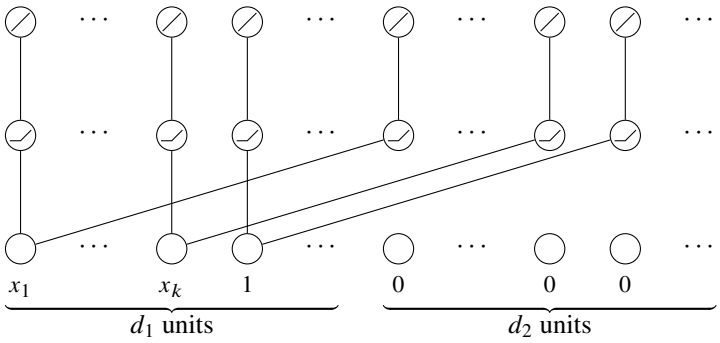
Case $c_0 + \sum_i c_i x_i = 0$: Same as above, but with the following FFN.



Case $\neg\chi_1$: Since we are using a different representation of truth values, this is different from Lemma 6.



Case $\chi_1 \wedge \chi_2$: By the induction hypothesis, there are stacks of transformer layers of widths d_1 and d_2, respectively, that compute χ_1 and χ_2. Concatenate them by Lemma 7. Add a layer at the *bottom* whose self-attention does nothing and whose FFN copies the input from the first half to the second half:



Finally, add a layer on top whose self-attention does nothing and whose FFN computes the minimum of the outputs of the two halves, as in the conjunction case of Lemma 6.

Case $\chi_1 \vee \chi_2$: Same as the previous case, but compute the maximum instead of the minimum, as in the disjunction case of Lemma 6.

D. Layer Normalization

Layer normalization shifts and scales a vector to have some learned mean and standard deviation (Ba et al., 2016). In this appendix, we give a brief definition of layer normalization as it relates to our other definitions, and describe how to modify the proof of Theorems 2 and 5 to take layer normalization into account.

D.1. Definition

Definition 16. *Layer normalization* with width d is a function

$$\begin{aligned} \mathrm{LN}\colon \mathbb{R}^d &\to \mathbb{R}^d \\ [\mathrm{LN}(x)]_i &= \gamma_i \frac{x_i - \bar{x}}{\sqrt{\mathrm{Var}(x) + \epsilon}} + \beta_i \end{aligned}$$

where

$$\begin{aligned} \bar{x} &= \frac{1}{d} \sum_i x_i \\ \mathrm{Var}(x) &= \frac{1}{d} \sum_i (x_i - \bar{x})^2 \end{aligned}$$

and $\gamma, \beta \in \mathbb{R}^d$ are learned. If $A \in \mathbb{R}^{d \times n'}$, we can write $\mathrm{LN}(A)$ by analogy with Eq. (3).

The term ϵ is added in all implementations we are aware of for numerical stability, although the original definition (Ba et al., 2016) has $\epsilon = 0$.

Then the equations for a transformer layer (Definition 5) are modified to:

$$\begin{aligned} \mathrm{Layer}\colon \mathbb{R}^{d \times n'} &\to \mathbb{R}^{d \times n'} \\ A &\mapsto A'' \text{ where} \\ A' &= \mathrm{LN}^{(1)}\left(\sum_{h=1}^{H} \mathrm{SA}^{(h)}(A) + A\right) \\ A'' &= \mathrm{LN}^{(2)}\left(\mathrm{FF}(A') + A'\right) \end{aligned}$$

where $\mathrm{LN}^{(1)}$ and $\mathrm{LN}^{(2)}$ are layer normalizations. Although some variants place layer normalization before each layer (Wang et al., 2019; Nguyen & Salazar, 2019), we follow the original definition, which places layer normalization after the residual connection.

D.2. Modified proof of Theorem 2

In §5, we justified the use of limited-precision, limited-range numbers by the fact that in a transformer whose activation functions are all Lipschitz continuous, the activations are bounded (Hahn, 2020). However, layer normalization with $\epsilon = 0$ is not Lipschitz continuous. Fortunately, we can show that activations are bounded even with $\epsilon = 0$.

Proposition 21. *For any transformer encoder with layer normalization with $\epsilon = 0$, there exists r such that, for all ℓ, i, and p, if $A_{ip}^{(\ell)}$ exists then $|A_{ip}^{(\ell)}| < 2^r$.*

Proof. For any $x \in \mathbb{R}^d$, let $z_i = \frac{x_i - \bar{x}}{\sqrt{\mathrm{Var}(x)}}$. Then

$$\frac{1}{z_i^2} = \frac{\sum_{j=1}^{d}(x_j - \bar{x})^2}{d(x_i - \bar{x})^2} \geq \frac{1}{d}$$

so $|z_i| \leq \sqrt{d}$, and layer normalization is bounded. The fact that all other sublayers are continuous implies that all activations are also bounded. □

As for the proof of Theorem 2 itself, defining layer normalization in $\mathsf{FOC}[+;\mathsf{MOD}]$ is straightforward by Proposition 3.

D.3. Modified proof of Theorem 5

In §6, we proved Theorem 5 without layer normalization. Adding layer normalization complicates the construction somewhat.

In the first step (computing the ψ_i), we modify the encoding of truth values so that layer normalization has no effect. Instead of representing a truth value or a count using a single activation, we use a pair of activations (following Chiang & Cholak (2022)). In the first step (§6.1), we use the pair $(1, 0)$ for true and $(0, 1)$ for false. This makes it possible to guarantee that activation matrices have the following property.

Definition 17. We say that a matrix $A \in \mathbb{R}^{d \times n'}$ has *row-mean* μ and *row-variance* σ if $A_{:,p}$ has mean μ and variance σ for all p. Then a function $f\colon \Sigma^* \to \mathbb{R}^{d \times n'}$ is *self-normalizing* if $f(w)$ has row-mean and row-variance not depending on w, and a function $f\colon \mathbb{R}^{d \times n'} \to \mathbb{R}^{d \times n'}$ is self-normalizing if $f(A)$ has row-mean and row-variance depending only on A's row-mean and row-variance.

Self-normalization makes it possible to set the parameters of layer normalization so that it has no effect.

Proposition 22. *If $f\colon \mathbb{R}^{d \times n'} \to \mathbb{R}^{d \times n'}$ is self-normalizing then there exist β, γ such that $LN(f(A); \beta, \gamma) = f(A)$.*

It is easy to modify the proof of Lemma 6 for the new representation of truth values and to guarantee that the resulting transformer encoder is self-normalizing. In particular, Lemma 7 constructs a self-normalizing $T_1 \oplus T_2$, provided T_1 and T_2 are self-normalizing.

The second step (computing counting quantifiers) produces values of the form $\frac{x_i}{n'}$, which we now modify to produce pairs of values $(\frac{x_i}{n'}, -\frac{x_i}{n'})$ to guarantee that activation matrices have a row-mean of zero. We can no longer guarantee that activations matrices have known row-variance, so layer normalization will rescale activations.

Consequently, in the third step (§6.3), we use $(+\delta, -\delta)$ for true and $(-\delta, +\delta)$ for false, where δ can be any positive number. It is easy to show that Lemma 8 still holds.