ENCODER–DECODER TRANSFORMERS: LOGICAL CHARACTERIZATIONS AND PERIODICITY

Anonymous authors
Paper under double-blind review

ABSTRACT

We give logical characterizations of encoder–decoder transformers, the foundational architecture for LLMs that also sees use in various settings that benefit from cross-attention, in the practical setting of floating-point numbers and soft attention. First, we characterize such transformers via a new temporal logic that extends propositional logic with a counting global modality over the encoder input and a past modality over the decoder input, as well as via a type of distributed automata. We consider three frameworks: with and without a final softmax step in the transformer, and in the setting where each model generates tokens via autoregression. Second, we show that both autoregressive transformers and sentences of counting propositional logic—the fragment of the previous logic obtained by omitting the past modality—recognize exactly the commutative star-free languages. Finally, we find that the sequences of tokens the transformers generate are ultimately periodic (and each token appears in the period at most once). This allows us to characterize autoregressive transformers via sentences of counting propositional logic that generate tokens without autoregression, i.e., we can effectively eliminate recursion from the transformers.

1 INTRODUCTION

Transformers (Vaswani et al., 2017) are the dominant architecture in modern natural language processing. They take words as inputs and incorporate various attention mechanisms and neural networks to produce numeric representations of the meanings of words (or more precisely, components of words called tokens). These representations are iteratively refined and used to sample from a selection of tokens to generate text. Due to the ubiquity of transformers, their expressive power has become an active area of research, with a rich body of results connecting to formal language theory, temporal logic, and circuit complexity (Strobl et al., 2024).

So far, nearly all such results have concerned either encoder-only transformers (using unmasked self-attention) or decoder-only transformers (using masked self-attention). The original encoder–decoder architecture of Vaswani et al. (2017) that combines unmasked and masked self-attention with cross-attention has been mostly absent, with the sole exception of the Turing-completeness construction of Pérez et al. (2021); notably, no fine-grained characterization exists. This gap is significant as encoder–decoder models remain standard for various settings, such as machine translation (NLLB Team, 2024) and connecting distinct modalities like text, audio, and video via cross-attention (Zheng et al., 2024; Sekoyan et al., 2025).

In this paper, we fill this gap with a detailed investigation into the formal expressivity of floating-point soft-attention encoder–decoder transformers.[1] We first give novel logical characterizations at multiple stages of transformer output via a new temporal logic and a class of distributed automata. While these results are of independent interest, they are also necessary stepping stones toward a logical characterization of autoregressive encoder–decoder transformers in the language recognition setting. They also lead to an interesting observation about the periodicity of floating-point transformers and a subsequent characterization of autoregressive transformers—in terms of generated token sequences—that essentially eliminates autoregression.

[1] We consider transformers without positional encodings, the study of which we leave for future work. Our results hold for some variations in the transformer architecture regarding whether attention layers have multiple heads, the strictness of the masking and the inclusion of layer normalization; see Appendix F.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In Section 3.1, we characterize transformers that simply output feature vectors, i.e., vectors of floats; this is a fundamental building block toward later results. In particular, we show that in this setting encoder–decoder transformers are characterized by the logic GPTL^- which extends propositional logic with a counting global modality over the encoder input and a past modality over the decoder input. We also show that they are expressively equivalent to a class of distributed automata called counting past-global distributed automata (CPG-automata) where the state transition function depends on a vertex's own previous state and two bounded multisets of previous states, one attending to vertices in the encoder input and the other attending to prior vertices in the decoder input.

Theorem 3. *Encoder–decoder transformers without the final* $\mathrm{softmax}$*,* GPTL^- *and* CPG*-automata have the same expressive power.*

In practice, transformers output a probability distribution over the available tokens via the $\mathrm{softmax}$ function. We give a characterization in this setting in Section 3.2. Since $\mathrm{softmax}$ cannot output every bit string, we use a similarity relation $\sim$ which intuitively connects nearby output vectors together; objects are equivalent if they produce similar outputs whenever possible.

Theorem 4. *For each similarity relation* $\sim$*, encoder–decoder transformers with the final* $\mathrm{softmax}$*,* GPTL^- *and* CPG*-automata have the same expressive power w.r.t.* $\sim$*.*

The purpose of the probability distribution is to enable autoregressive token generation; we study this setting in Section 3.3. Here, transformers, GPTL^- and CPG-automata work in a periodic fashion to generate an output string from an input string. The output string is initialised as a beginning-of-string token. Iteratively, the input and the so-far generated output are fed to the transformer (respectively, tuple of GPTL^- or CPG-automaton), after which a token is sampled based on the output using a function that maps bit strings to tokens. The sampled token is appended to the end of the output, and the process repeats until an end-of-string token is produced. Two objects are generation-equivalent if they generate the same outputs with the same inputs.

Theorem 5. *As autoregressive token-generating systems, encoder–decoder transformers,* GPTL^- *and* CPG*-automata are equally expressive regardless of if the transformers use the final* $\mathrm{softmax}$ *or not.*

With these characterizations in hand, we study autoregressive transformers further. In Section 4.1, we first investigate the setting of language recognition; a word is fed to the encoder and is accepted iff an accepting token is generated at any point. We show that the commutative star-free languages are exactly those recognized by transformers, counting propositional logic CPL (the fragment of GPTL^- without the past modality) and k-capped multiset functions, which accept inputs based on how many (up to the bound k) of each token they have.

Theorem 6. *Sentences of CPL, accepting encoder–decoder transformers and* k*-capped multiset functions recognize exactly the commutative star-free languages.*

A natural follow-up is to ask how much of the autoregression we can capture via CPL. Theorem 5 reveals a useful insight into answering this question: transformers are ultimately periodic, meaning that if the generated output sequence is infinite, it eventually begins looping indefinitely, with each loop containing at most one of each token.

Proposition 7. *Each encoder–decoder transformer is ultimately periodic and the periods are repetition-free.*

The culmination of this work in Section 4.2 involves using this insight to characterize the *entire token sequences* generated by autoregressive transformers in terms of sentences of CPL arranged into two matrices, the first corresponding to the prefix of the sequence and the second to the repeating period.

Theorem 8. *As token-generating systems, autoregressive encoder–decoder transformers and sentences of CPL (without autoregression) are equally expressive regardless of if the transformers use the final* $\mathrm{softmax}$ *or not.*

We conclude the work and discuss limitations in Section 5. Our results constitute a major step forward in the formal study of encoder–decoder transformers. Our result that transformers are ultimately periodic also naturally raises many questions regarding the more general relationship between autoregression and expressivity, which we leave for future work.

Related work. The expressive power of fixed-precision transformers has received much attention in recent years. We focus here on the work directly relevant to this paper; for more, see Appendix A.

The papers closest to our work are Li & Cotterell (2025), Yang et al. (2026) and Ahvonen et al. (2026b). We first give an overview of their results and then contrast them with our work. Li & Cotterell (2025) showed that floating-point decoder-only transformers recognize exactly the left-deterministic polynomial languages and characterized such transformers via the past-only fragment of linear temporal logic (LTL[P]) as well as via partially ordered deterministic finite automata. Yang et al. (2026) extended this characterization (as well as corresponding ones for real-valued transformers and unique hard-attention) to the language modeling setting where equivalence is over probability distributions. Ahvonen et al. (2026b) characterized floating-point encoder-only transformers via propositional logic extended with a counting global modality; this also applies to transformers over graphs.

We emphasize our contributions in contrast to the above papers. First, while the mentioned papers concern encoder-only and decoder-only transformers, we characterize encoder–decoder transformers which combine the two via cross-attention in a non-trivial way. Our characterizations are via two different logics and a new type of distributed automata, and they reveal that encoder–decoder transformers recognize exactly the commutative star-free languages. Second, our language recognition framework works natively via token generation unlike the common approach in the literature adopted in Li & Cotterell (2025) where transformers are treated as binary classifiers directly; we also show that these two frameworks are equivalent. Finally, our work also goes in an entirely new direction: we observe that the token sequences produced by floating-point transformers are ultimately periodic and the resulting periods contain at most one of each token. We use this to give a whole new type of result where an autoregressive transformer is characterized via a logic that is not used autoregressively.

Regarding other characterizations of decoder-only transformers, Yang et al. (2024) showed that over reals masked rightmost-hard-attention transformers recognize exactly the star-free languages, establishing an equivalence with linear temporal logic. For fixed precision, Jerad et al. (2025) used LTL[P] to capture leftmost-hard attention transformers. Yang et al. (2025) characterized transformers that round to fixed precision except inside attention via LTL[P] extended with counting operators. We show that soft-attention floating-point encoder–decoder transformers recognize exactly the commutative star-free languages, establishing them as a less expressive variant than the transformers in Yang et al. (2024), Li & Cotterell (2025), Jerad et al. (2025) and Yang et al. (2025).

To our knowledge, the sole previous work on the formal expressivity of encoder–decoder transformers is Pérez et al. (2021). They consider autoregressive transformers in terms of language recognition and show that they are Turing-complete with rational numbers of arbitrary precision, average-hard attention and positional encodings. They also note that without positional encodings or with fixed precision such transformers cannot separate words with the same ratios of tokens. By contrast, most of our results are more general than language recognition, and in the case of language recognition we give a precise characterization of floating-point encoder–decoder transformers with soft attention and no positional encodings via the commutative star-free languages. This reveals orthogonal expressivity between arbitrary precision and floats in the framework of encoder–decoder transformers and no positional encodings (not taking into account the subtle differences between soft and average hard-attention): with floats they cannot recognize certain properties recognizable with arbitrary precision such as "there are more tokens a than tokens b", but they can recognize some properties not recognizable with arbitrary precision such as "there are precisely five tokens a". This is consistent with prior observations from Ahvonen et al. (2026b).

2 PRELIMINARIES

Let $\mathbb{N} = \{0, 1, \dots\}$ and $\mathbb{Z}_+ = \{1, 2, \dots\}$. For all $n \in \mathbb{Z}_+$, let $[n] := \{1, \dots, n\}$. We denote finite sets of **tokens** by Σ. We often denote strings and sequences by bold letters $\mathbf{s}$ or arrows $\vec{s}$. A **multiset over** S is a function $S \to \mathbb{N}$, and a k**-capped multiset over** S is a function $S \to \{0, \dots, k\}$. We denote the set of multisets over S as $\mathcal{M}(S)$ and the set of k-capped multisets over S as $\mathcal{M}_k(S)$.

2.1 WORD-SHAPED GRAPHS

Transformers operate on words which we represent as graphs. A Σ**-labeled word-shaped graph** $\mathcal{W}$ is a tuple (V, E, λ) where $V = [n]$ $(n \in \mathbb{Z}_+)$ is a set of **vertices**, E is the successor relation $\{(i, i+1) \mid i \in [n-1]\}$ (inducing a strict linear order $<$ over V), and $\lambda : V \to \Sigma$ is a **labelling function** assigning a token to each vertex. We may use $V(\mathcal{W})$ to denote V. A **pointed** Σ**-labeled word-shaped graph** is a pair $(\mathcal{W}, i)$ where $i \in V(\mathcal{W})$.

In particular, a transformer operates on two words, one fed to the encoder and the other to the decoder. We represent the two inputs as a two-sorted graph. A Σ**-labeled two-sorted word-shaped graph** is a $\Sigma \cup \{\mathrm{BOS}\}$-

labeled graph $\mathcal{W}$ where exactly one vertex i has the token BOS ("Beginning Of String"). The **prefix** and the **suffix** are the subgraphs induced by vertices $1, \ldots, i-1$ and $i, \ldots, n$ respectively. In the autoregressive setting, we will also call the prefix the **input** and the suffix the **output**. For simplicity, we will refer to word-shaped graphs as words. We also refer to token strings as words, but the meaning is always clear from the context.

2.2 Floating-point numbers

We study encoder–decoder transformers with fixed numerical precision. Each transformer uses a single floating-point format for all calculations, but different transformers may use different formats.

Given $p, q \in \mathbb{Z}_+$, a **floating-point number** (or **float**) over p and q is a bit string $b_0 \cdots b_{q+p} \in \{0,1\}^{p+q+1}$. The bit b_0 encodes the **sign**, $\mathbf{e} := b_1 \cdots b_q$ is the **exponent**, representing $e \in \mathbb{N}$ in binary, and $b_{q+1} \cdots b_{q+p}$ is the **significand**, representing $s \in \mathbb{N}$ in binary. Let $a = 2^{p-1}$ and $b = 2^{q-1} - 1$. Except for special cases described below, the float $b_0 \cdots b_{q+p}$ encodes the real number $(-1)^{b_0} \frac{s}{a} 2^{e-b}$. If $\mathbf{s} = 0^p$ and $\mathbf{e} = 1^q$, the float represents ∞ for $b_0 = 0$ and $-\infty$ for $b_0 = 1$. When unambiguous, we identify a float with the real value (or $\pm\infty$) it represents.

A float is called **normalized** if $b_{q+1} = 1$, and **subnormalized** if $b_{q+1} = 0$ and $\mathbf{e} = 0^q$. The **floating-point format** $\mathcal{F}(p, q)$ consists of all normalized and subnormalized floats over p and q, together with ∞, $-\infty$, and a designated float for NaN (Not a Number). When p and q are clear from the context, we simply write $\mathcal{F}$.

Arithmetic operations $+$, $-$, $\cdot$, $\div$, and $\sqrt{\ }$ over $\mathcal{F}(p, q)$ are defined as follows. First compute the exact result r in extended real arithmetic, and then map it to a float in $\mathcal{F}(p, q)$ as follows. If r is undefined (e.g., ∞/∞) or an operand is NaN, the result is NaN. If r is representable in $\mathcal{F}(p, q+1)$ but not in $\mathcal{F}(p, q)$, then r exceeds the range of $\mathcal{F}(p, q)$ and the result is $\pm\infty$ according to its sign. Otherwise, r is rounded to the nearest float in $\mathcal{F}(p, q)$. Special functions such as $\exp(x)$ are approximated within the float format, e.g., using Taylor or related series.

2.3 Encoder–decoder transformers

Our encoder–decoder transformer is modeled after the original architecture of Vaswani et al. (2017); see Figure 1. In this section, we give an intuitive overview of this architecture; for detailed definitions, see Appendix B, and for variations on masking, multi-head attention and layer normalization, see Appendix F.

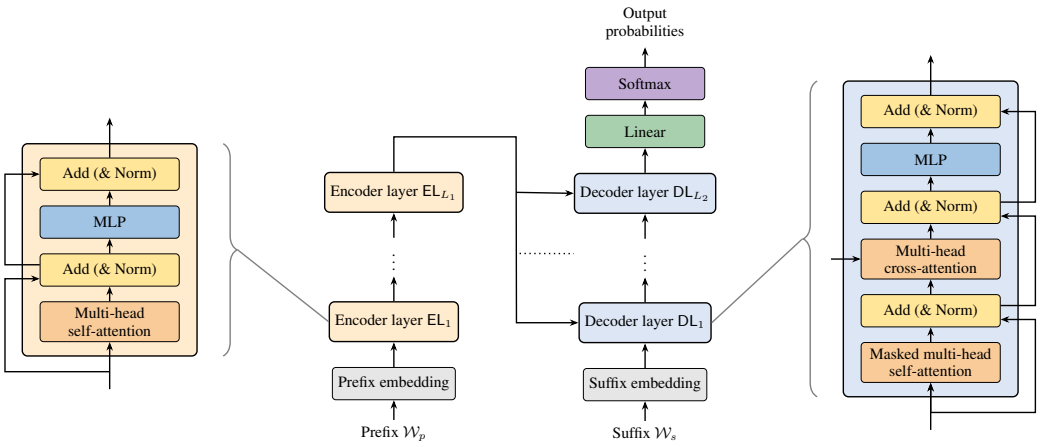


Figure 1: The encoder–decoder transformer of Vaswani et al. (2017) with positional encodings omitted. Layer normalization ("Norm") is included for reference but does not affect expressivity; see Appendix F.3.

Our framework is floating-point numbers, soft attention, strict causal masking and no positional encodings. An **encoder–decoder transformer** with dimension d has two parts: an encoder operating on prefixes $\mathcal{W}_p$ and a decoder operating on suffixes $\mathcal{W}_s$. First an **embedding function** $\Sigma \to \mathcal{F}^d$ is applied vertex-wise. The encoder then applies a stack of **encoder layers**, each consisting of a self-attention sub-layer followed by an MLP. The decoder applies a stack of **decoder layers**, each consisting of a masked self-attention sub-layer, a cross-attention sub-layer and an MLP. Each sub-layer and MLP has input and output dimension d and is wrapped in a **residual connection** that adds its input to its output. MLPs have two layers and use $\mathrm{ReLU}(x) = \max\{0, x\}$ in the first.

All attention sub-layers are built from **attention heads**

$$H(X,Y) := \mathrm{softmax}\Big(\frac{(YW^Q)(XW^K)^\top}{\sqrt{d_k}}\Big)(XW^V),$$

where W^Q, W^K, W^V are weight matrices, d_k is a hidden dimension and $\mathrm{softmax}$ is applied row-wise. A **self-attention head** is $H(X) := H(X,X)$. A **cross-attention head** $H(X,Y)$ combines the output X of the encoder with the output Y of the previous sub-layer of the decoder, so each decoder position attends to the whole encoder output. In **masked** self-attention, a **strict causal mask** sets the entries on and above the diagonal to $-\infty$ before $\mathrm{softmax}$, so positions attend only to strictly prior positions. Attention sub-layers may apply many attention heads of the same kind in parallel; the outputs are concatenated followed by a linear mapping to return to dimension d.

The function softmax is a length-preserving function on float vectors: for $\mathbf{x} = (x_1, \ldots, x_n)$ the ith component of $\mathrm{softmax}(\mathbf{x})$ is defined as $\frac{e^{x_i - \max(\mathbf{x})}}{\sum_{j=1}^{n} e^{x_j - \max(\mathbf{x})}}$. The denominator contains a sum of floats, but the sum of floats is not associative. For example, in the format $\mathcal{F}(3,3)$ we have $(1+4)+5 = 5+5 = 10$ whereas $1+(4+5) = 1+8 = 8$ because 9 is not representable in $\mathcal{F}(3,3)$ and rounds to 8. Thus, we need to fix an order for summation. Summing in the order of the vertices would allow the output of $\mathrm{softmax}$ in a particular position to depend on how the values in the other positions are ordered, which is not desirable. We thus instead adopt the convention to always sum in the increasing order of floats, which is a reasonable choice w.r.t. accuracy (Higham, 1993; Robertazzi & Schwartz, 1988; Wilkinson, 1959). This leads to the sum of a multiset of floats being bounded in the sense that it only depends on the multiplicities of floats up to some bound k. Formally, for a multiset M of floats, let $\mathrm{SUM}_{\mathcal{F}}(M)$ denote the value of the sum in which each float f appears $M(f)$ times and the summands are added in increasing order, and let $M_{|k}$ denote the k**-projection** of M, i.e., $M_{|k}(x) := \min\{M(x), k\}$. Then we obtain:

Proposition 1 (Ahvonen et al., 2024)**.** *For each floating-point format $\mathcal{F}$, there exists a $k \in \mathbb{N}$ such that for all multisets M over floats in $\mathcal{F}$, we have $\mathrm{SUM}_{\mathcal{F}}(M) = \mathrm{SUM}_{\mathcal{F}}(M_{|k})$.*

There are three stages in which transformers produce outputs. In the first stage, the output of the decoder is linearly mapped into a desired **output dimension** d_{out}, so every suffix vertex ends up with a vector in $\mathcal{F}^{d_{\text{out}}}$; we study this stage in Section 3.1. A final $\mathrm{softmax}$ can then be applied, typically with $d_{\text{out}} = |\Sigma \cup \{\mathrm{EOS}\}|$ (where EOS is a special "End Of String"-token) to obtain a probability distribution over tokens; we examine this stage in Section 3.2. Finally, this can be used to autoregressively generate tokens; we study this in Sections 3.3 and 4.

2.4 Logic

The first logic we consider is **two-sorted global past temporal logic** (GPTL^-).[2] The Σ**-formulae of** GPTL^- are constructed according to the following grammar:

$$\varphi ::= \bot \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \langle G \rangle^{\text{pre}}_{\geq k}\, \varphi \mid \langle P \rangle^{\text{suf}}\, \varphi,$$

where $p \in \Sigma$ and $k \in \mathbb{N}$. The semantics of GPTL^- is as follows. Given a Σ-formula φ of GPTL^- and a pointed Σ-labeled two-sorted word $(\mathcal{W}, i)$: **(1)** $\mathcal{W}, i \models \bot$ never holds, **(2)** $\mathcal{W}, i \models p$ iff $\lambda(i) = p$, **(3)** the semantics of $\neg$ and $\wedge$ are defined as usual, **(4)** $\mathcal{W}, i \models \langle G \rangle^{\text{pre}}_{\geq k}\, \varphi$ iff $|\{j \in V(\mathcal{W}_p) \mid \mathcal{W}, j \models \varphi\}| \geq k$, and **(5)** $\mathcal{W}, i \models \langle P \rangle^{\text{suf}}\, \varphi$ iff $\exists j \in V(\mathcal{W}_s)$ such that $j < i$ and $\mathcal{W}, j \models \varphi$. For a tuple $\vec{\varphi} = (\varphi_1, \ldots, \varphi_m)$ of formulae of GPTL^- and $i \in V(\mathcal{W}_s)$, the **output of** $\vec{\varphi}$ **at** $(\mathcal{W}, i)$ is the bit string $b_1 \cdots b_m$ such that $b_\ell = 1$ iff $\mathcal{W}, i \models \varphi_\ell$.

Counting propositional logic (CPL) is the fragment of GPTL^- that does not contain the past modality $\langle P \rangle^{\text{suf}}$. A formula of CPL is a **sentence** if all instances of proposition symbols occur under the scope of a modality $\langle G \rangle^{\text{pre}}_{\geq k}$. We evaluate sentences of CPL over words in the natural way: a sentence is true iff it is true in any vertex.

2.5 Automata

We also consider a type of distributed automata operating on two-sorted words where vertices receive messages from all prefix vertices and preceding suffix vertices. Formally, a **counting past-global distributed automaton** (CPG-automaton) over the vocabulary Σ is a tuple $A = (Q, \pi, \delta, n, b)$ consisting of a non-empty set $Q \subseteq \{0,1\}^m$ ($m \in \mathbb{Z}_+$) of **states**, an **initializing function** $\pi : \Sigma \to Q$ (assigning an initial state to a vertex based on its token), a **transition function** $\delta\colon Q \times \mathcal{M}_k(Q) \times \mathcal{M}_k(Q) \to Q$, an **output round** $n \in \mathbb{N}$ and **output length** $b \in \mathbb{Z}_+$.

[2] The minus in GPTL^- refers to the restriction of the modalities to the prefix and suffix respectively.

At each synchronous step, each vertex updates its state based on: **(1)** its current state, **(2)** the k-capped multiset of current states of prefix vertices, and **(3)** the k-capped multiset of current states of prior suffix vertices. More formally, we define the state x_i^t of a vertex i of a two-sorted word $\mathcal{W} = (V, E, \lambda)$ after t iterations of A as follows. For $t = 0$, $x_i^0 := \pi(\lambda(i))$. Next, assume x_j^t is defined for all $j \in V$. Let i_0 be the one vertex labeled BOS. Then

$$x_i^{t+1} := \delta(x_i^t, \{\{\, x_j^t \mid j < i_0 \,\}\}_{|k}, \{\{\, x_j^t \mid i_0 \leq j < i \,\}\}_{|k}).$$

The vertex updates stop in round n, i.e., after n iterations of δ. The prefix of length b of x_i^n (with i in the suffix) is called the **output of** A **at** $(\mathcal{W}, i)$. Intuitively, the truncation means that the output does not have to contain the auxiliary information used in internal computation; for example, if $b = 1$ then the output is binary: "yes" or "no".

3 Characterizing encoder–decoder transformers via GPTL^-

We begin our analysis by giving logical characterizations of encoder–decoder transformers in three output stages. We start with the fundamental case where the transformers can have any output dimension but no final softmax. Building on this, we then consider the case where they do use a final softmax and output a probability distribution over tokens. Finally, we extend even further to the case where the transformers generate tokens via autoregression. While these results are of independent interest, they are also a necessary step toward a more sophisticated characterization of encoder–decoder transformers which we will consider in Section 4.

3.1 Equivalence before the final softmax

We first compare transformers, logic and automata at the level of the pre-$\mathrm{softmax}$ output (the **logits**). Let A and B be classes consisting of encoder–decoder transformers, tuples of formulae of GPTL^- and CPG-automata, and let $a \in A$ and $b \in B$. We say that a and b are **equivalent** (or **logit-equivalent**) if they produce the same outputs with the same inputs. There are two ways to interpret the final feature vector $(\mathbf{f}_1, \ldots, \mathbf{f}_{d_{\mathrm{out}}}) \in \mathcal{F}^{d_{\mathrm{out}}}$ (representing the rational numbers $f_1, \ldots, f_{d_{\mathrm{out}}}$) of a given vertex as a bit string. The first is simply $\mathbf{f}_1 \cdots \mathbf{f}_{d_{\mathrm{out}}}$ which we use by default. Only in this section (3.1), when translating from logic or automata to transformers, we construct the transformer such that $f_1, \ldots, f_{d_{\mathrm{out}}} \in \{0, 1\}$ and instead interpret the output as $f_1 \cdots f_{d_{\mathrm{out}}}$. We say that A and B have the same expressive power if for each $a \in A$ there exists an equivalent $b \in B$ and vice versa.

Translating from logic to transformers utilises a float phenomenon called **underflow**: any number as close or closer to zero than to any other float in the format rounds to zero. The following proposition exemplifies this.

Proposition 2 (Ahvonen et al., 2026b)**.** *Let $\mathcal{F}$ be a floating-point format, f the smallest positive float in $\mathcal{F}$, and k an integer such that $\frac{k}{2}$ is accurately representable in $\mathcal{F}$. Then, for all $F \in \mathcal{F}$, $|F| \leq |\frac{1}{k}|$ iff $F \cdot (\frac{k}{2} \cdot f) = 0$.*

Our first characterization theorem is as follows.

Theorem 3. *Encoder–decoder transformers without the final* $\mathrm{softmax}$*,* GPTL^- *and* CPG*-automata have the same expressive power.*

Proof. (Sketch.) (Logic $\Rightarrow$ transformer.) The truth value of each subformula is computed in its own feature column. Connectives $\neg$ and $\wedge$ are local to a vertex and handled by MLPs. Each $\langle G \rangle_{\geq k}^{\mathrm{pre}}$ is handled by an unmasked self-attention or cross-attention head constructed following Ahvonen et al. (2026b), i.e., we use Proposition 2 so that underflow occurs when k or more prefix vertices satisfy the subformula. Each $\langle P \rangle^{\mathrm{suf}}$ is handled by a masked self-attention head. The final linear map extracts the relevant truth values. Details are in Appendix C.1.

(Transformer $\Rightarrow$ automaton.) Each attention sub-layer and each MLP is simulated by a transition of the automaton. MLPs are local, so the transition function simply acts as a look-up table. Unmasked self-attention and cross-attention are computed from the first multiset of the transition function and masked self-attention from the second. By Proposition 1, the sums in an attention sub-layer depend on multiplicities only up to the saturation bound k of $\mathcal{F}$, so k-capped multisets suffice. Details are in Appendix C.2.

(Automaton $\Rightarrow$ logic.) For each state q of the automaton A, we construct a formula φ_q^n that holds at a pointed two-sorted word $(\mathcal{W}, i)$ iff A assigns q to i in the output round n. The formula φ_q^n is the disjunction of the **types** of all pointed words that receive q, where the type of $(\mathcal{W}, i)$ is a formula describing i precisely enough that every vertex satisfying it receives the same states as i in the first n rounds of every CPG-automaton. The formula for the mth output bit is the disjunction of those φ_q^n where the mth bit of q is 1. Details are in Appendix C.3. □

3.2 EQUIVALENCE AFTER THE FINAL SOFTMAX

In the transformer architecture of Vaswani et al. (2017), the final linear transformation has output dimension $|\Sigma \cup \{\text{EOS}\}|$, after which a final softmax layer is applied to generate a probability distribution over the output vocabulary at each position. We extend our characterization to this level via equivalence up to a notion of *similarity*, since not all outputs of formula tuples and automata are possible outputs of the $\mathrm{softmax}$ function.

Fix a **similarity relation** $\sim \subseteq \mathcal{F}^{d_{\text{out}}} \times \mathcal{F}^{d_{\text{out}}}$.[3] Intuitively, $\sim$ could connect feature vectors based on how close the values of the vectors are to each other, but we allow any relation $\sim$. Let A and B be classes consisting of encoder–decoder transformers with the final $\mathrm{softmax}$, tuples of formulae of GPTL^- and CPG-automata, and let $a \in A$ and $b \in B$. We say that a is **equivalent to** b **w.r.t.** $\sim$ if for each two-sorted word $\mathcal{W}$ and each vertex i in the suffix of $\mathcal{W}$, the following holds: if the output of a at $(\mathcal{W}, i)$ is $\mathbf{v}$ and there exists an output $\mathbf{u}$ in the range of b such that $\mathbf{v} \sim \mathbf{u}$, then b outputs such an output $\mathbf{u}$ at $(\mathcal{W}, i)$. We say that A and B **have the same expressive power w.r.t.** $\sim$ if for each $a \in A$ there exists some $b \in B$ that is equivalent to a w.r.t. $\sim$ and vice versa.

Theorem 4. *For each similarity relation $\sim$, encoder–decoder transformers with the final* $\mathrm{softmax}$*,* GPTL^- *and* CPG*-automata have the same expressive power w.r.t. $\sim$.*

Proof. We extend the translations of Theorem 3 as follows. From transformers to automata, the automaton first computes the pre-$\mathrm{softmax}$ output $\mathbf{b}$ exactly as before, and then applies a fixed map on bit strings that sends $\mathbf{b}$ to some output similar to $\mathrm{softmax}(\mathbf{b})$ whenever one exists. From automata to logic, we first construct a tuple that simulates the output of the automaton as before; these formulae can then be used to map to a similar output instead by using the connectives $\neg$ and $\wedge$. From logic to transformers, we compute the truth values of subformulae as before, which contains the output $\mathbf{b}$ of the formula tuple. We add extra decoder layers where the MLPs together with the final linear map implement a look-up table that, whenever $\mathbf{b}$ is similar to some output of $\mathrm{softmax}$, produces a pre-image of that output; the final $\mathrm{softmax}$ then yields an output similar to $\mathbf{b}$. □

3.3 GENERATION-EQUIVALENCE

Moving beyond the final softmax, we extend our characterization to the token generation framework by extending encoder–decoder transformers, tuples of GPTL^- and CPG-automata to be autoregressive. A **token-generating system** is a pair (a, f), where a is one the above objects over Σ and $f : \{0,1\}^n \to \Sigma \cup \{\text{EOS}\}$ where n is the length of the output bit strings of a.[4] Let $\mathcal{W}$ be a word over Σ. First we turn $\mathcal{W}$ into a two-sorted word $\mathcal{W}^{(1)}$ by adding a vertex labeled BOS to the end. Tokens are generated by, at step $t > 1$, computing the output o of a at the last vertex of $\mathcal{W}^{(t-1)}$ and extending $\mathcal{W}^{(t-1)}$ by a vertex labeled $f(o)$ to obtain $\mathcal{W}^{(t)}$. The loop terminates if $f(o) = \text{EOS}$. The (possibly infinite) sequence of tokens generated is the **output sequence** of (a, f) on $\mathcal{W}$. Two token-generating systems are **generation-equivalent** if they produce the same output sequences with the same inputs. We say that two classes A and B of token-generating systems **have the same expressive power** if for each $a \in A$ there exists a generation-equivalent $b \in B$ and vice versa.

Theorem 5. *As autoregressive token-generating systems, encoder–decoder transformers,* GPTL^- *and* CPG*-automata are equally expressive regardless of if the transformers use the final* $\mathrm{softmax}$ *or not.*

Proof. We extend the translations of Theorem 3 as follows. When translating between logic and automata or away from transformers, we simply use the same token-generating function f. From logic or automata to transformers, we construct an injection i that maps the possible output strings of the token-generating system (a, f) into corresponding outputs of $\mathrm{softmax}$ and then choose a token-generating function f' such that $f(\mathbf{b}) = (f' \circ i)(\mathbf{b})$ for each possible output $\mathbf{b} \in \{0,1\}^n$ of (a, f). We present one possible way of constructing i and leave possible improvements for future work. First, we note that each $\mathbf{b}$ encodes a natural number $b \leq 2^n - 1$ in binary. We add a constant number of decoder layers to the transformer constructed in the translation of Theorem 3; together with the final linear transformation, the MLPs of the added layers compute the encoding $e : \{0,1\}^n \to \mathcal{F}^{|\Sigma \cup \{\text{EOS}\}|}$ such that $e(\mathbf{b}) = (b, 0, \ldots, 0)$. We choose a large enough float format $\mathcal{F}$ such that **(1)** each b is expressible in $\mathcal{F}$, and **(2)** $\mathrm{softmax}$ is injective with respect to the image of e. The injection i that we seek is $\mathrm{softmax} \circ e$. □

[3] Note that $\sim$ can be generalised to a relation between all bit strings of the same length as those in $\mathcal{F}^{d_{\text{out}}}$ by interpreting missing strings via the corresponding normalized and subnormalized floats in $\mathcal{F}$.

[4] Here the definition of f allows only deterministic token generation. By changing the range of f to be a power set or a set of probability distributions over $\Sigma \cup \{\text{EOS}\}$, we could allow non-deterministic or probabilistic token generation; we leave this mostly for future work but state that Theorems 5 and 8 still hold as the proofs do not depend on the range of f.

4 ELIMINATING AUTOREGRESSION

The logical characterizations in Section 3 make it possible to obtain logical characterizations of autoregressive transformers via logics that are not autoregressive. We first consider the case where an autoregressive transformer is treated as a language recognizer. Building on this, we then examine the general case where we characterize the exact token sequences generated by the transformer.

4.1 LANGUAGE RECOGNITION

We continue our analysis by considering the autoregressive encoder–decoder transformers studied in Section 3.3 in the special case of language recognition. This is a natural framework especially in the case of encoder–decoder transformers as the input is fed into the encoder only. An **accepting** encoder–decoder transformer is one with a distinguished acceptance token. It **accepts** an input if this token is generated at any point; otherwise it **rejects**.

As a useful tool, we define a very simple object that also recognizes languages. For $k \in \mathbb{Z}_+$, the **k-capped multiset of a Σ-labeled word** $\mathcal{W}$ is the multiset $M_k^{\mathcal{W}}$ that counts the number of each token in $\mathcal{W}$ up to the bound k. A **k-capped multiset function** is a function $f\colon \mathcal{M}_k(\Sigma) \to \{0,1\}$ evaluated on $\mathcal{W}$ as $f(M_k^{\mathcal{W}})$.

Given a language L of (finite) words over Σ, we say that an accepting encoder–decoder transformer **recognizes** L if it accepts exactly the words in L, a CPL-sentence recognizes L if it is true in exactly the words in L, and a k-capped multiset function recognizes L if it maps exactly the words in L to 1.

Our result is a four-way characterization of the languages recognized by transformers. It shows that accepting transformers, sentences of CPL and k-capped multiset functions all recognize exactly the **commutative star-free** languages, i.e., permutation-invariant languages definable via regular expressions with complement but without the Kleene star.

Theorem 6. *Sentences of CPL, accepting encoder–decoder transformers and k-capped multiset functions recognize exactly the commutative star-free languages.*

Proof. (Sketch.) The details of all directions are in Appendix D. **(1)** From CPL to transformers, we modify the translation from the proof of Theorem 3 such that the transformer outputs an accepting token if and only if the formula is true. **(2)** From transformers to multiset functions, we first translate the transformer into a CPG-automaton via Theorem 5 and observe that the only information gained from the input is the k-capped multiset of the input. **(3)** From multiset functions to CPL, it is easy to construct for each multiset in $\mathcal{M}_k(\Sigma)$ a corresponding sentence of CPL, and a disjunction over multisets that are mapped to 1 gives us the desired sentence. **(4)** To show that a language recognized by a multiset function is commutative and star-free, we construct a suitable regular expression that defines the language. **(5)** To recognize a commutative star-free language with a multiset function, we utilize Schützenberger's theorem (Schützenberger, 1965). □

The commutative star-free languages are a strict subset of the left-deterministic polynomial languages recognized by decoder-only transformers and characterized by LTL[P] (Li & Cotterell, 2025). Thus, our result implies that encoder–decoder transformers are in fact less expressive than decoder-only transformers. For example, the language consisting of the words where b does not precede a is not permutation-invariant and thus not recognizable by any encoder–decoder transformer, but it is definable by a formula of LTL[P] and thus recognizable by a decoder-only transformer. This is essentially due to the fact that, in our framework, the input word is fed to the encoder rather than the decoder.

Note that Theorem 6 is not dependent on the fact that our accepting transformers are autoregressive. We could also treat the encoder–decoder transformer as a binary classifier without autoregression, e.g. such that the output dimension is 1 and the transformer accepts iff it outputs a non-negative float (Yang et al., 2024; Li & Cotterell, 2025). This can be simulated by a sentence of CPL by replacing the autoregressive transformers and automata in the proof of Theorem 6 with the non-autoregressive kind, and the reverse holds by Theorem 3.

4.2 TOKEN GENERATION

We now know we can express the existence of an accepting token in the generated sequence with a single sentence of CPL. This begs the question: could we express the entire output sequence in a similar way? Remarkably, the answer is yes. Building on Theorem 5, we first discover that transformers are **ultimately periodic**, meaning that

for each input, either the output is finite or there exists a point at which the tokens generated begin repeating periodically. While this phenomenon is not specific to transformers—consider ultimately periodic languages and automata (Calbrix et al., 1993; Bresolin et al., 2009), and, e.g., MSO-automata (Walukiewicz, 1996)—in our case, the periods generated are **repetition-free**, i.e., each token appears at most once per period.

Proposition 7. *Each encoder–decoder transformer is ultimately periodic and the periods are repetition-free.*

Proof. (Sketch.) The proof utilizes the equivalence of transformers and GPTL^- in Theorem 5. We show that a formula φ (or a sequence of formulae) of GPTL^- cannot distinguish suffix vertices that are labeled with the same token and agree on which subformulae of φ are satisfied by prior vertices in the suffix. Thus, since the set of subformulae of φ that are true in some prior suffix vertex is non-decreasing, eventually the next token to be generated depends only on the last generated token. For the details, see Appendix D. □

Proposition 7 implies that the translations in Theorem 6 are computable. It also means that each individual output sequence can be encoded in finite space. Thus, since Theorem 6 shows that a transformer can only distinguish a finite number of inputs, we can encode all generated output sequences over all inputs in finite space.

Let Ψ_1 and Ψ_2 be matrices of CPL-sentences[5] with n rows and let $f : \{0,1\}^n \to \Sigma \cup \{\mathrm{EOS}\}$. Evaluating the sentences over an input word $\mathcal{W}$ yields binary matrices $\Psi_1(\mathcal{W})$ and $\Psi_2(\mathcal{W})$, and applying f to their columns from left to right yields token strings $\vec{\sigma}_1$ and $\vec{\sigma}_2$. Let $\vec{\sigma}_2'$ be the longest prefix of $\vec{\sigma}_2$ that does not contain any token twice. The output of (Ψ_1, Ψ_2) with input $\mathcal{W}$ is then the infinite string that starts with $\vec{\sigma}_1$ and is followed by infinitely many copies of $\vec{\sigma}_2'$, cut off to be finite after the first EOS if it contains one.

Theorem 8. *As token-generating systems, autoregressive encoder–decoder transformers and sentences of CPL (without autoregression) are equally expressive regardless of if the transformers use the final* $\mathrm{softmax}$ *or not.*

Proof. (Sketch.) From transformers to CPL, each column of sentences reads the input $\mathcal{W}$ and produces a bit string that is a pre-image of the token generated by the token-generating function of the transformer in that position. This is possible since the token to be generated can be inferred from the input and there is a finite number of inputs that the transformer can distinguish. From CPL to transformers, we first use decoder layers to track which column of the first matrix to simulate; once the matrix is finished, we only need to look at the current token to do the same with the second matrix indefinitely. For the details, see Appendix E. □

5 Conclusion

We have given novel logical characterizations of encoder–decoder transformers via a new temporal logic and a new type of distributed automata. The characterizations cover three different output frameworks: with and without the final softmax step of the transformer as well as autoregressive token generation. We have then used these characterizations to obtain further results for autoregressive transformers. In terms of language recognition, we have shown that they recognize exactly the commutative star-free languages and are characterized by counting propositional logic. We have also shown that transformer outputs are ultimately periodic and each token appears at most once per period. Using this, we have characterized the exact autoregressive behaviour of the transformers via a finitary logic that is neither autoregressive nor recursive.

Finally, we stress that with minimal adjustments, our translations could cover decoder-only transformers as a special case. With an empty prefix, the counting global modality becomes vacuous and GPTL^- collapses to the logic LTL[P] by which Li & Cotterell (2025) characterize fixed-precision decoder-only transformers. We especially note that autoregressive decoder-only transformers are also ultimately periodic by a similar argument.

Limitations. Our work is purely theoretical, and no experiments were performed. We only considered the case without positional encodings; it would be interesting to also characterize both the sinusoidal positional encodings of Vaswani et al. (2017) and more modern choices like rotary positional embeddings (RoPE) (Su et al., 2024). Finally, there are alternative ways to treat floats which we did not investigate, like allowing a single transformer to operate with various float formats depending on the input.

[5]The CPL-sentences could of course be replaced by k-capped multiset functions, accepting encoder–decoder transformers or expressions defining commutative star-free languages, and Theorem 8 would still hold.

AI USE DISCLOSURE

In this work, we used generative AI tools to aid writing (e.g. rewording, condensing, proposing alternate versions of paragraphs) and for retrieving related work. We have not used generative AI tools for research ideation, to draft entire sections of the paper or to prove mathematical claims, and generating synthetic datasets are not applicable to this work. The TikZ code for Figure 1 was initially generated by AI and then manually adjusted by the authors. We have reviewed all AI-assisted work. We have checked all references and read the related work suggested by the AI, and have manually proof-read and edited all text generated by the AI. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical Characterizations of Recurrent Graph Neural Networks with Reals and Floats. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 104205–104249. Curran Associates, Inc., 2024. doi: 10.52202/079017-3311. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/bca7a9a0dd85e2a68420e5cae27eccfb-Paper-Conference.pdf.

Veeti Ahvonen, Maurice Funk, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Expressive Power of Graph Transformers via Logic, 2026a. URL https://arxiv.org/abs/2508.01067.

Veeti Ahvonen, Maurice Funk, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Expressive Power of Graph Transformers via Logic. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(24):19569–19579, Mar. 2026b. doi: 10.1609/aaai.v40i24.39036. URL https://ojs.aaai.org/index.php/AAAI/article/view/39036.

Pablo Barcelo, Alexander Kozachinskiy, Anthony W. Lin, and Vladimir Podolskii. Logical Languages Accepted by Transformer Encoders with Hard Attention. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 22077–22087, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5f0fdc1acd47431f7f3bb8ee85598cef-Paper-Conference.pdf.

Davide Bresolin, Angelo Montanari, and Gabriele Puppis. A theory of ultimately periodic languages and automata with an application to time granularity. *Acta Informatica*, 46(5):331–360, 2009.

Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω-languages. In *International Conference on Mathematical Foundations of Programming Semantics*, pp. 554–566. Springer, 1993.

David Chiang and Peter Cholak. Overcoming a Theoretical Limitation of Self-Attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7654–7664, 2022.

David Chiang, Peter Cholak, and Anand Pillay. Tighter Bounds on the Expressivity of Transformer Encoders. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Nicholas J. Higham. The Accuracy of Floating Point Summation. *SIAM J. Sci. Comput.*, 14(4):783–799, 1993. doi: 10.1137/0914050.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 977–996, 2025.

Hongjian Jiang, Michael Hahn, Georg Zetzsche, and Anthony Widjaja Lin. Softmax Transformers are Turing-Complete, 2025. URL https://arxiv.org/abs/2511.20038.

Antti Kuusisto. Modal Logic and Distributed Message Passing Automata. In Simona Ronchi Della Rocca (ed.), *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 452–468, Dagstuhl, Germany, 2013. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-939897-60-6. doi: 10.4230/LIPIcs.CSL.2013.452. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2013.452.

Jiaoda Li and Ryan Cotterell. Characterizing the Expressivity of Fixed-precision Transformer Language Models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 159510–159554. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/e9e250537b0345111d50a5f8f392cffc-Paper-Conference.pdf.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of Thought Empowers Transformers to Solve Inherently Serial Problems. In *The Twelfth International Conference on Learning Representations*, 2024.

Robert McNaughton and Seymour A Papert. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

William Merrill and Ashish Sabharwal. A Logic for Expressing Log-Precision Transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 52453–52463. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a48e5877c7bf86a513950ab23b360498-Paper-Conference.pdf.

William Merrill and Ashish Sabharwal. The Expressive Power of Transformers with Chain of Thought. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/1f59721c106ea80f613299039112f651-Paper-Conference.pdf.

NLLB Team. Scaling neural machine translation to 200 languages. *Nature*, 630(8018):841–846, 2024.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is Turing Complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021. URL http://jmlr.org/papers/v22/20-302.html.

Thomas G. Robertazzi and Stuart C. Schwartz. Best "Ordering" for Floating-Point Addition. *ACM Trans. Math. Softw.*, 14(1):101–110, 1988. doi: 10.1145/42288.42343.

M.P. Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *Information and Control*, 8(2):190–194, 1965. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(65)90108-7. URL https://www.sciencedirect.com/science/article/pii/S0019995865901087.

Monica Sekoyan, Nithin Rao Koluguri, Nune Tadevosyan, Piotr Zelasko, Travis Bartley, Nikolay Karpov, Jagadeesh Balam, and Boris Ginsburg. Canary-1b-v2 & parakeet-tdt-0.6 b-v3: Efficient and high-performance models for multilingual asr and ast. *arXiv preprint arXiv:2509.14128*, 2025.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What Formal Languages Can Transformers Express? A Survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024. ISSN 2307-387X. doi: 10.1162/tacl_a_00663. URL http://dx.doi.org/10.1162/tacl_a_00663.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *Neurocomputing*, 568:127063, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Igor Walukiewicz. Monadic second order logic on tree-like structures. In Claude Puech and Rüdiger Reischuk (eds.), *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pp. 401–413. Springer, 1996. doi: 10.1007/3-540-60922-9_33.

James Hardy Wilkinson. Rounding Errors in Algebraic Processes. In *Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO, Paris 15-20 June 1959*, pp. 44–53. UNESCO (Paris), 1959.

Andy Yang and David Chiang. Counting Like Transformers: Compiling Temporal Counting Logic Into Softmax Transformers. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FmhPg4UJ9K.

Andy Yang, David Chiang, and Dana Angluin. Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 10202–10235. Curran Associates, Inc., 2024. doi: 10.52202/079017-0327. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/13d7f172259b11b230cc5da8768abc5f-Paper-Conference.pdf.

Andy Yang, Anej Svete, Jiaoda Li, Anthony W Lin, Jonathan Rawski, Ryan Cotterell, and David Chiang. Probability distributions computed by autoregressive transformers. In *International Conference on Learning Representations*, volume 2026, pp. 88582–88601, 2026.

Andy J Yang, Michaël Cadilhac, and David Chiang. Knee-Deep in C-RASP: A Transformer Depth Hierarchy. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 37184–37225. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/354a6e7be4f1a35127746b8147a31bfe-Paper-Conference.pdf.

Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. Open-Sora: Democratizing Efficient Video Production for All, 2024. URL https://arxiv.org/abs/2412.20404.

A Additional related work

In this section, we cover related work pertaining to upper and lower bounds on the expressivity of transformers as well as chain-of-thought models.

Regarding lower and upper bounds for the expressive power of transformers, Chiang et al. (2023) established $\mathsf{FOC}[+;\mathsf{MOD}]$ as an upper bound for fixed-precision encoder-only transformers and as a lower bound for the case with reals. Merrill & Sabharwal (2023) showed that first-order logic with majority quantifiers is an upper bound for log-precision transformers. Yang & Chiang (2024) established Minimal Tense Logic with counting terms as a lower bound for masked soft-attention transformers with unbounded input size. Barcelo et al. (2024) gave first-order logic with unary numerical predicates as a lower bound for unique hard-attention transformers with reals as well as linear temporal logic with unary numerical predicates and counting formulae as a lower bound for average hard-attention.

Another direction has been to investigate the effect of *chain-of-thought* (CoT) on decoder-only models, i.e., allowing the model to generate "scratchpad" tokens that are discarded before the final answer. Merrill & Sabharwal (2024) began in the log-precision setting, showing that CoT increases expressive power depending on the number of intermediate steps, a polynomial number capturing exactly the complexity class P. Li et al. (2024) showed that with finite-precision and a constant amount of decoder layers, CoT allows constant-depth transformers to solve inherently serial problems that would otherwise require proportionally many layers. Jiang et al. (2025) showed that length-generalizable soft-attention transformers with CoT are Turing-complete, where length-generalizability means that an idealized learning procedure will converge to the language expressible with the transformer model when given all words of some length in the language.

B Detailed definitions for encoder–decoder transformers

In this section, we give a more detailed definition of encoder–decoder transformers.

First, some notations. For a matrix X, we let $X_{i,*}$ denote the ith row, $X_{*,j}$ the jth column, and $X_{i,j}$ the element in the ith row and jth column of X. For a set S, let S^* denote the set of finite sequences over S, let $S^{*\times *}$ denote the set of matrices over S and let $S^{m\times *}$ and $S^{*\times n}$ denote the restriction of $S^{*\times *}$ to matrices with m rows and n columns, respectively. For matrices $X^{(1)},\dots,X^{(h)} \in S^{m\times n}$, we let $\mathrm{concat}(X^{(1)},\dots,X^{(h)})$ denote the matrix Y where $Y_{i,(\ell-1)n+j} = X^{(\ell)}_{i,j}$.

We define an encoder–decoder transformer over a floating-point format $\mathcal{F}$ as follows.

Embedding. First, an **embedding function** of dimension d is a function $\mathrm{em} : \Sigma \to \mathcal{F}^d$ that maps each token to a vector in the embedding space. Applying it vertex-wise to the prefix $\mathcal{W}_p$ and suffix $\mathcal{W}_s$ thus produces matrices in $\mathcal{F}^{|\mathcal{W}_p|\times d}$ and $\mathcal{F}^{|\mathcal{W}_s|\times d}$, where the ith row of the former is $\mathrm{em}(\lambda(i))$ and the jth row of the latter is $\mathrm{em}(\lambda(|\mathcal{W}_p|+j))$.

Masking. The **strict causal mask** maps each square matrix $S \in \mathcal{F}^{*\times *}$ to the matrix $\mathrm{mask}(S)$ of the same dimensions with the upper-right-triangular entries set to $-\infty$. In other words, $\mathrm{mask}(S)_{ij} := -\infty$ if $j \geq i$ and $\mathrm{mask}(S)_{ij} := S_{ij}$ otherwise. For non-strict masking and the shifting operation of Vaswani et al. (2017), see Appendix F.2.

Softmax. The softmax function intuitively turns each vector into a probability distribution. More formally, for each vector $\mathbf{x} = (x_1,\dots,x_n) \in \mathcal{F}^*$, $\mathrm{softmax}(\mathbf{x}) \in \mathcal{F}^*$ is the vector of the same length as $\mathbf{x}$ such that $\mathrm{softmax}(\mathbf{x})_i := \frac{e^{x_i-\max(\mathbf{x})}}{\sum_{j=1}^{n} e^{x_j-\max(\mathbf{x})}}$ where the bias $\max(\mathbf{x})$ is subtracted for numerical stability.

Attention. An **attention head** with dimensions (d, d_k, d_v) consists of parameters $W^Q, W^K \in \mathcal{F}^{d\times d_k}$ and $W^V \in \mathcal{F}^{d\times d_v}$, and maps a pair of matrices $X, Y \in \mathcal{F}^{*\times d}$ to a matrix with as many rows as Y defined as

$$H_{\mathrm{mask}}(X,Y) := \mathrm{softmax}\Big(\mathrm{mask}\Big(\frac{(YW^Q)(XW^K)^\top}{\sqrt{d_k}}\Big)\Big)(XW^V) \in \mathcal{F}^{*\times d_v},$$

where $\mathrm{softmax}$ is applied row-wise. Intuitively, X and Y are preceding outputs obtained from the encoder and decoder respectively. We consider three special cases of attention heads: the **masked self-attention head**

$H_{\mathrm{mask}}(Y) := H_{\mathrm{mask}}(Y, Y)$[6], and two cases where masking is omitted: the **cross-attention head** $H(X, Y)$ and the **unmasked self-attention head** $H(X) := H(X, X)$.

A **multi-head attention layer of width** h combines heads $H^{(1)}, \ldots, H^{(h)}$ of one of the above kinds with dimensions (d, d_k, d_v) with a parameter matrix $W^O \in \mathcal{F}^{hd_v \times d}$. For unmasked self-attention:

$$\mathsf{MHSA}(X) := \mathrm{concat}(H^{(1)}(X), \ldots, H^{(h)}(X))\, W^O \in \mathcal{F}^{* \times d},$$

and analogously $\mathsf{MHSA}(Y)_{\mathrm{mask}}$ for masked self-attention and $\mathsf{MHCA}(X, Y)$ for cross-attention; the output has d columns and as many rows as X (resp. Y). Multiple heads do not increase expressivity (see Appendix F.1), but makes some proofs easier.

MLPs. An **MLP** (multi-layer perceptron) with dimensions (d, d_{ff}) is a function $M : \mathcal{F}^{1 \times d} \to \mathcal{F}^{1 \times d}$ defined w.r.t. matrices $W_1 \in \mathcal{F}^{d \times d_{ff}}$ and $W_2 \in \mathcal{F}^{d_{ff} \times d}$ and bias vectors $b_1 \in \mathcal{F}^{1 \times d_{ff}}$ and $b_2 \in \mathcal{F}^{1 \times d}$. For each (horizontal) vector $\mathbf{x} \in \mathcal{F}^{1 \times d}$, we define

$$M(\mathbf{x}) := \mathrm{ReLU}(\mathbf{x}W_1 + b_1)W_2 + b_2,$$

where $\mathrm{ReLU}(x) = \max\{0, x\}$ is applied element-wise. For a matrix $X \in \mathcal{F}^{* \times d}$, $M(X)$ is obtained by applying M separately to each row of X.

Layers. An **encoder layer** EL and a **decoder layer** DL with dimensions (d, d_k, d_v, d_{ff}) combine attention layers and MLPs as follows where the inputs and outputs are in $\mathcal{F}^{* \times d}$, $\mathsf{EL}(X)$ has as many rows as X and $\mathsf{DL}(X, Y)$ has as many rows as Y:

$$\mathsf{EL}(X) := M(X') + X' \qquad \text{where } X' := \mathsf{MHSA}(X) + X,$$
$$\mathsf{DL}(X, Y) := M(Y'') + Y'' \qquad \text{where } Y'' := \mathsf{MHCA}(X, Y') + Y' \text{ and } Y' := \mathsf{MHSA}_{\mathrm{mask}}(Y) + Y,$$

each sub-layer having its own parameters. The sums are called **residual connections**. Layer normalization, commonly applied before or after the residual connections, does not affect expressivity and is omitted (see Appendix F.3). An **encoder** E and **decoder** D **of length** L are compositions of L such layers:

$$\mathsf{E}(X) := (\mathsf{EL}_L \circ \cdots \circ \mathsf{EL}_1)(X),$$
$$\mathsf{D}(X, Y) := \mathsf{DL}_L(X, \mathsf{DL}_{L-1}(X, \cdots \mathsf{DL}_2(X, \mathsf{DL}_1(X, Y)) \cdots)).$$

Linear transformation. Given the final decoder output $Y \in \mathcal{F}^{|\mathcal{W}_s| \times d}$ and a fixed output dimension d_{out}, a final **linear transformation** with dimensions (d, d_{out}) with parameter $W^{\text{out}} \in \mathcal{F}^{d \times d_{\text{out}}}$ maps the output to

$$\mathrm{Out}(Y) := Y \cdot W^{\text{out}} \in \mathcal{F}^{|\mathcal{W}_s| \times d_{\text{out}}}.$$

Encoder–decoder transformer. An **encoder–decoder transformer** T with dimensions $(d, d_k, d_v, d_{ff}, d_{\text{out}})$ consists of an embedding function em of dimension d, an encoder E and decoder D of dimensions (d, d_k, d_v, d_{ff}), and a linear transformation Out with dimensions (d, d_{out}). T maps each Σ-labeled two-sorted word $\mathcal{W}$ to

$$\mathrm{T}(\mathcal{W}) := \mathrm{Out}(\mathsf{D}(\mathsf{E}(\mathrm{em}(\mathcal{W}_p)), \mathrm{em}(\mathcal{W}_s))) \in \mathcal{F}^{|\mathcal{W}_s| \times d_{\text{out}}}.$$

At every vertex v in the suffix $\mathcal{W}_s$, an encoder–decoder transformer T outputs a vector $\mathbf{v} = (f_1, \ldots, f_{d_{\text{out}}}) \in \mathcal{F}^{d_{\text{out}}}$. This can be interpreted as a bit string in two natural ways.

- **Bit-wise interpretation:** We can always interpret $\mathbf{v}$ as the concatenated bit string $f_1 \cdots f_{d_{\text{out}}}$.
- **Feature-wise interpretation:** If f_i represents either 0 or 1 for each $i \in [d_{\text{out}}]$, then we can also interpret $\mathbf{v}$ as the bit string $b_1 \cdots b_{d_{\text{out}}}$ where $b_i = 1$ if and only if f_i represents 1.

Let a be either a tuple of GPTL^--formulae or a CPG-automaton, and let T be an encoder–decoder transformer.

- We say that T is **feature-wise equivalent** to a if for each pointed two-sorted graph $(\mathcal{W}, i)$ the output of T at $(\mathcal{W}, i)$ interpreted feature-wise is the same as the output of a at $(\mathcal{W}, i)$.
- We say that a is **bit-wise equivalent** to T if for each pointed two-sorted graph $(\mathcal{W}, i)$ the output of a at $(\mathcal{W}, i)$ is the same as the output of T at $(\mathcal{W}, i)$ interpreted bit-wise.

[6] After masking and before $\mathrm{softmax}$, the top row is $(-\infty, \ldots, -\infty)$. If $\mathrm{softmax}$ is computed normally, this results in a row of NaNs. To avoid this, we adopt the convention that $\mathrm{softmax}(-\infty, \ldots, -\infty) := (0, \ldots, 0)$. This is a natural choice that simply stops the BOS-token from making predictions about future tokens.

C PROOFS FOR THEOREM 3

In this section, we prove Theorem 3. We split the proof into three parts: in each part, we prove one of the following three statements.

Theorem 9 (Logic ⇒ transformers)**.** *For each tuple of formulae of* GPTL^{-}*, we can construct a feature-wise equivalent encoder–decoder transformer without the final* $\mathrm{softmax}$*.*

Theorem 10 (Transformers ⇒ automata)**.** *For each floating-point encoder–decoder transformer without the final* $\mathrm{softmax}$*, we can construct a bit-wise equivalent* CPG*-automaton.*

Theorem 11 (Automata ⇒ logic)**.** *For each* CPG*-automaton, we can construct an equivalent tuple of formulae of* GPTL^{-}*.*

C.1 PROOF OF THEOREM 9

In this section, we prove Theorem 9.

Theorem 9 (Logic ⇒ transformers)**.** *For each tuple of formulae of* GPTL^{-}*, we can construct a feature-wise equivalent encoder–decoder transformer without the final* $\mathrm{softmax}$*.*

Proof. We intuitively use the MLPs to simulate Boolean operators, unmasked self-attention layers and/or cross-attention layers to simulate modalities $\langle G \rangle^{\mathrm{pre}}_{\geq k}$ and masked self-attention layers to simulate modalities $\langle P \rangle^{\mathrm{suf}}$.

Let $\vec{\varphi} = (\varphi_1, \ldots, \varphi_m)$ be a tuple of formulae of GPTL^{-}. Let $\psi_1, \ldots, \psi_d$ enumerate the subformulae of $\varphi_1, \ldots, \varphi_m$ including all tokens and $\top$ and $\bot$ such that $\psi_i = \varphi_i$ for each $i \in [m]$. We basically use d as the internal dimension of our transformer, but there is no harm in padding the internal dimension up to, say, an integer multiple of d to avoid problems resulting from residual connections. This can be used to ensure that the inputs and outputs of sub-layers do not interfere with each other by shifting the output to a different section of the feature dimension while unnecessary input values can be negated by the MLPs; the details of shifting are already covered in Ahvonen et al. (2026a). For the embedding function, we map each token p_i to the feature vector in $\mathcal{F}^d$ such that the feature in position j is 1 if $\psi_j \in \{p_i, \top\}$ and 0 otherwise. We then move to compute the truth values of subformulae other than tokens, $\top$ and $\bot$, one-by-one. The method of computing the truth value will depend on the connective used to construct the subformula in question.

Simulating Boolean operators: For Boolean operators $\neg$ and $\wedge$, we use the MLPs in decoder layers. We set all weights in all attention sub-layers to zero; because of the residual connection, this means that these attention sub-layers only achieve an identity transformation. For the MLPs, we recall from Lemma 38 of Ahvonen et al. (2026a) that a ReLU-activated MLP with arbitrarily many layers can simulate Boolean operators in the sense that columns of the feature matrix correspond to subformulae and the truth values of more complex subformulae are computed into their own columns based on the columns corresponding to their immediate subformulae. Thus, a stack of encoder or decoder layers with two-layer MLPs can do the same.

Simulating $\langle P \rangle^{\mathrm{suf}}$: We use masked self-attention sub-layers to simulate $\langle P \rangle^{\mathrm{suf}}$ as follows. Assume that column i encodes the truth values of φ; we want to calculate the truth value of $\langle P \rangle^{\mathrm{suf}}\varphi$. Let W^Q and W^K be $(d \times 1)$-matrices where all elements are 0s. Now $\frac{(XW^Q)(XW^K)^{\intercal}}{\sqrt{d_k}}$ is the $n \times n$ matrix where every element is 0.

After masking, every element on and to the right of the diagonal becomes $-\infty$. Biasing does not affect any of the rows. After $\mathrm{softmax}$, all values to the left of the diagonal are non-zero while all the values to the right of the diagonal are 0. Once we define W^V as the $(d \times 1)$-matrix where the ith row is 1 and others are 0s, the attention head outputs an $(n \times 1)$-matrix where each row is either some non-zero number if $\langle P \rangle^{\mathrm{suf}}\varphi$ is true in the corresponding vertex and 0 if $\langle P \rangle^{\mathrm{suf}}\varphi$ is false instead. We can then use the MLPs in the decoder layers to normalize all non-zero values to 1 while keeping all 0s as 0s; see Lemma 40 in Ahvonen et al. (2026a).

Simulating $\langle G \rangle^{\mathrm{pre}}_{\geq k}$: It was already shown in Theorem 47 in Ahvonen et al. (2026a) how a stack of encoder layers can simulate $\langle G \rangle^{\mathrm{pre}}_{\geq k}$. However, we also need to be able to simulate them with a stack of decoder layers

since modalities $\langle G \rangle^{\text{pre}}_{\geq k}$ can be stacked on top of $\langle P \rangle^{\text{suf}}$; since we used decoder layers to simulate $\langle P \rangle^{\text{suf}}$, this means that the decoder layers must be able to handle both $\langle G \rangle^{\text{pre}}_{\geq k}$ and $\langle P \rangle^{\text{suf}}$.

Assume that X and Y are the Boolean input matrices to the cross-attention sub-layer where the ith column encodes the truth value of φ and the jth column encodes $\top$ (i.e., it is a column of 1s). The matrix W^Q has a single column where the jth row is F and others are zero, where F is the greatest non-infinite float in the floating-point format. Thus, YW^Q is a single-column matrix of Fs. The matrix W^K has likewise a single column where the ith row is 1 and others are 0. Hence, the matrix XW_K has a single column which encodes the truth values of φ. Now $(YW^Q)(XW^K)^\mathsf{T}$ is a matrix where each row is the ith column of X multiplied by F. This is otherwise the same outcome as in the construction from Ahvonen et al. (2026a;b), except there are no rows of zeros unless φ is true in zero vertices. This is strictly beneficial as it only eliminates the possibility of false positives. The rest of the construction follows the same scheme as Ahvonen et al. (2026a;b); the idea is to construct W_V in a simple way that triggers underflow (see Proposition 2) exactly when the number of vertices satisfying φ is at least k. This only works for half of the possible values of k but, as covered in Ahvonen et al. (2026a), the remaining cases can be covered with additional arithmetical analysis and the MLPs can be used to normalize all values back into 0s and 1s correctly.

The final linear transformation of the transformer simply projects the first m elements of the feature vectors, giving us the simulated truth values of $\varphi_1, \ldots, \varphi_m$. □

C.2 Proof of Theorem 10

In this section, we prove Theorem 10.

Theorem 10 (Transformers ⇒ automata)**.** *For each floating-point encoder–decoder transformer without the final* softmax*, we can construct a bit-wise equivalent* CPG*-automaton.*

Proof. We construct a CPG-automaton that simulates the whole transformer such that we only need to perform a single state transition per attention sub-layer and MLP. All the internal arithmetic is essentially encoded into the transition function.

Let T be an encoder–decoder transformer of dimension d over the float format $\mathcal{F}(p, q)$ with N_1 encoder layers and N_2 decoder layers. We construct a bit-wise equivalent CPG-automaton $(Q, \pi, \delta, 2N_1 + 3N_2 + 1, d)$ where $Q = \{0,1\}^{d(p+q+1)+m}$ where the first $d(p+q+1)$ bits are used to simulate T while the last m bits (with a suitably large choice of m) encode which sub-layer of which encoder or decoder layer we are simulating: $0 \cdots 0$ for the self-attention sub-layer of the first encoder layer, $0 \cdots 01$ for the MLP of the first encoder layer, $0 \cdots 010$ for the next attention sub-layer, etc.

Let $\mathrm{em} : \Sigma \to \mathcal{F}^d$ be the embedding function of the transformer T. This can be identified with a function $f : \Sigma \to \{0,1\}^{d(p+q+1)}$ where the first $p+q+1$ bits encode the first float in the output of em, the next $p+q+1$ bits encode the second one, and so on. We define that $\pi(p_i)$ is $\mathrm{em}(p_i)$ concatenated with the string $0 \cdots 0$, indicating that the calculation begins with an unmasked self-attention head.

Finally, we define the transition function; we do this based on which sub-layer we are simulating. For the remainder of the proof, we will omit the last m bits of the states of the automaton, since the last m bits simply tick upward in a lexicographic order. We will further simplify notation by writing $q_{\mathbf{v}}$ to denote the state corresponding to the feature vector $\mathbf{v} \in \mathcal{F}^d$. We also simplify the setting with one more core observation; each vertex can tell whether it is in the encoder-part or decoder-part of the word based on the size of the second multiset in the transition function; thus, it is easy to define that during encoder layers, decoder vertices do not update, and during decoder layers, encoder vertices do not update.

Simulating unmasked self-attention: We analyse the arithmetic operations in the attention sub-layer and analyse the dependencies to show that a single transition suffices to simulate the whole layer.

Assume for each vertex i and each feature vector $\mathbf{v}$ that i is in state $q_{\mathbf{v}}$ if and only if i has the feature vector $\mathbf{v}$ before applying the self-attention module. We focus on the simpler case where there is only one attention head (i.e., $h = 1$) and each feature vector is a single floating-point number; the technique we use generalises easily to the case with more attention heads and longer feature vectors. Letting $\mathbf{v}_i$ denote the feature vector of vertex i (in this case a single float) before multiplication with W^Q and $\mathbf{v}_i^Q$ after, and letting Q_j denote the jth column of

W^Q (in this case a single float), we have

$$\mathbf{v}_i^Q = (\mathbf{v}_i \cdot Q_1, \ldots, \mathbf{v}_i \cdot Q_{d_k}).$$

The vectors $\mathbf{v}_i^K$ and $\mathbf{v}_i^V$ are calculated analogously, and each of them only depends on $\mathbf{v}_i$ (and not on any $\mathbf{v}_j$ where $j \neq i$).

Next, we compute F_{ij}^{QK}, which will be the float in row i and column j of $(XW^Q)(XW^K)^\mathsf{T}$. We get

$$F_{ij}^{QK} = \sum_{k=1}^{d_k} \mathbf{v}_{i,k}^Q \mathbf{v}_{j,k}^K.$$

Clearly, each F_{ij}^{QK} only depends on $\mathbf{v}_i^Q$ and $\mathbf{v}_j^K$.

Next, we compute $F_{ij}^{\sqrt{d_k}}$, which will be the float in row i and column j of $\frac{(XW^Q)(XW^K)^\mathsf{T}}{\sqrt{d_k}}$:

$$F_{ij}^{\sqrt{d_k}} = \frac{F_{ij}^{QK}}{\sqrt{d_k}}.$$

Next, we compute $F_{ij}^{\mathrm{softmax}}$, which will be the float in row i and column j of $\mathrm{softmax}\left(\frac{(XW^Q)(XW^K)^\mathsf{T}}{\sqrt{d_k}}\right)$. Let $i_{\max} := \max\{\, F_{ij}^{\sqrt{d_k}} \mid 1 \leq j \leq n \,\}$. We get

$$F_{ij}^{\mathrm{softmax}} = \frac{e^{F_{ij}^{\sqrt{d_k}} - i_{\max}}}{\sum_{k=1}^{n} e^{F_{ik}^{\sqrt{d_k}} - i_{\max}}}.$$

Due to Proposition 1, F_i^{softmax} depends only on $F_{ik}^{\sqrt{d_k}}$ for a bounded number of k-values.

Next, we compute F_{ij}, which will be the float in row i and column j of the output matrix of the attention head:

$$F_{ij} = \sum_{k=1}^{n} F_{ik}^{\mathrm{softmax}} \mathbf{v}_{k,j}^V.$$

Again, due to Proposition 1, each F_{ij} depends on products $F_{ik}^{\mathrm{softmax}} \mathbf{v}_{k,i}^V$ for only a bounded number of k-values.

Finally, we consider the final output matrix O of the attention module. We have

$$O_{ij} = \sum_{k=1}^{hd_v} F_{ik} W_{kj}^O.$$

Due to the number of attention heads h being fixed, O_{ij} only depends on F_{ik} for a bounded number of k-values.

To show that the automaton A can compute O_i for each i in a single step, we only need to show that O_i depends on $\mathbf{v}_j$ for only a bounded number of j-values. Clearly O_i only depends on F_i. Since F_{ij} depends on products $F_{ik}^{\mathrm{softmax}} \mathbf{v}_{k,i}^V$ for only a bounded number of k-values, so do F_i and thus O_i because the number of columns j is fixed.

For each float, there are only a finite number of pairs of floats that, when multiplied, will result in the float in question; thus, O_i depends on $F_{ik}^{\mathrm{softmax}}$ and $\mathbf{v}_{k,i}^V$ for only a bounded number of k-values.

Because $\mathbf{v}_{k,i}^V$ only depends on $\mathbf{v}_k$, we only need to further consider $F_{ik}^{\mathrm{softmax}}$. Because F_i^{softmax} depends only on $F_{ik}^{\sqrt{d_k}}$ for a bounded number of k-values (as stated earlier), the same is true for O_i. Because $F_{ik}^{\sqrt{d_k}}$ only depends on F_{ik}^{QK}, O_i only depends on F_{ik}^{QK} for a bounded number of k-values. Finally, since F_{ik}^{QK} only depends on $\mathbf{v}_i^Q$ and $\mathbf{v}_k^K$ which only depend on $\mathbf{v}_i$ and $\mathbf{v}_k$ respectively, O_i only depends on $\mathbf{v}_k$ for a bounded number of k-values. Thus, the automaton A can calculate the transition for each vertex in a single step by only looking at the multiset of states $q_{\mathbf{v}}$ of vertices in the word. Moreover, since the residual connection does not add any dependencies, we can skip past O_i and calculate the output of the residual connection in the single transition step.

Simulating MLP**s:** Because MLPs perform only local transformations, we construct the transition function to act as a look-up table for the outputs of the MLP.

Since the weight matrices and biases are constant, for each $\mathbf{v}$ there exists a unique feature vector $\mathbf{v}'$ that is the output of the MLP when given the input $\mathbf{v}$. Thus, we define $\delta(q_{\mathbf{v}}, M, M') = q_{\mathbf{v}'}$.

Simulating masked self-attention sub-layers: We modify the case for unmasked self-attention to show that in the case of masked self-attention, the transitions depend only on the feature vectors of a bounded number of preceding vertices in the suffix of the word (and not at all on vertices that follow, or on vertices in the prefix).

First, we replace each occurrence of X with Y, as masked self-attention occurs in decoder layers. The core difference occurs in the step where we compute $F_{ij}^{\mathrm{softmax}}$; in the particular case of masked self-attention, we have $i_{\max} := \max\{\, F_{ij}^{\sqrt{d_k}} \mid 1 \leq j \leq i \,\}$ and the denominator only sums up to the bound j rather than n, i.e., we have

$$F_{ij}^{\mathrm{softmax}} = \frac{e^{F_{ij}^{\sqrt{d_k}} - i_{\max}}}{\sum_{k=1}^{j} e^{F_{ik}^{\sqrt{d_k}} - i_{\max}}}$$

for all $j \leq i$ and $F_{ij}^{\mathrm{softmax}} = 0$ for all $j > i$.

The analysis on dependencies is the same as before with the following additional observations. Because $F_{ij}^{\mathrm{softmax}} = 0$ for all $j > i$, O_i depends on products $F_{ik}^{\mathrm{softmax}} \mathbf{v}_{k,i}^{V}$ for only a bounded number of values $k \leq i$, and it thus only depends on $\mathbf{v}_{k,i}^{V}$ for a bounded number of values $k \leq i$. Likewise, because $F_{ij}^{\mathrm{softmax}}$ only depends on $F_{ik}^{\sqrt{d_k}}$ for a bounded number of values $k \leq i$, so does O_i, and this upper limit i cascades in the analysis until we reach $\mathbf{v}_k^K$. Thus, O_i only depends on $\mathbf{v}'_k$ for a bounded number of values $k \leq i$ and not at all for values $k > i$, and so we can read the transition from the second multiset in the same way as we did with the first multiset in the prior case.

Simulating cross-attention sub-layers: We modify the case for unmasked self-attention to show that in the case of cross-attention, the transitions depend only on the feature vectors of a bounded number of vertices in the encoder part of the word (and not at all on other vertices in the decoder part). Naturally, we replace X with Y in the product XW_i^Q. The dependence of O_i on $\mathbf{v}_k^Q$ is only for the value $k = i$, and the transition function can read this from the first input. The rest of the analysis remains unchanged.

Simulating the final linear transformation: Similar to MLPs, the final linear transformation can be calculated in a single step just by looking at the previous state of a vertex. This concludes the construction. □

C.3 Proof of Theorem 11

In this section, we prove Theorem 11.

Theorem 11 (Automata ⇒ logic)**.** *For each* CPG*-automaton, we can construct an equivalent tuple of formulae of* GPTL^{-}.

We start by defining types. Intuitively, these are formulae of GPTL^{-} that specify a pointed word uniquely up to some modal depth and width. These are analogous to graded types in Ahvonen et al. (2024) and types in Kuusisto (2013).

Before defining types, we need some auxiliary definitions. We define the abbreviated formula $\langle P \rangle^{\mathrm{suf}}_{\geq k} \varphi$ recursively as follows. For $k = 1$, we define $\langle P \rangle^{\mathrm{suf}}_{\geq 1} \varphi := \langle P \rangle^{\mathrm{suf}} \varphi$. Assume we have defined $\langle P \rangle^{\mathrm{suf}}_{\geq k} \varphi$. We define

$$\langle P \rangle^{\mathrm{suf}}_{\geq k+1} \varphi := \langle P \rangle^{\mathrm{suf}} (\varphi \wedge \langle P \rangle^{\mathrm{suf}}_{\geq k} \varphi).$$

Let $\mathrm{GPTL}^{-}_{\geq}$ be the logic obtained from GPTL^{-} by replacing $\langle P \rangle^{\mathrm{suf}}$ in the grammar with $\langle P \rangle^{\mathrm{suf}}_{\geq k}$. More formally, a Σ-formula φ of $\mathrm{GPTL}^{-}_{\geq}$ is constructed according to the following grammar:

$$\varphi ::= \bot \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \langle G \rangle^{\mathrm{pre}}_{\geq k} \varphi \mid \langle P \rangle^{\mathrm{suf}}_{\geq k} \varphi,$$

where $p \in \Sigma$ and $k \in \mathbb{N}$.

Since the modality $\langle P \rangle^{\text{suf}}_{\geq k}$ can be defined as an abbreviation in GPTL^{-} and since $\langle P \rangle^{\text{suf}}_{\geq 1}$ has the same semantics as $\langle P \rangle^{\text{suf}}$, the two logics GPTL^{-} and $\mathrm{GPTL}^{-}_{\geq}$ have the same expressive power: each formula of GPTL^{-} can be translated into a formula of $\mathrm{GPTL}^{-}_{\geq}$ and vice versa.

The **modal depth** $\mathsf{md}(\varphi)$ and **width** $\mathsf{width}(\varphi)$ of a formula φ of $\mathrm{GPTL}^{-}_{\geq}$ are defined recursively as follows:

$$
\begin{array}{rclrcl}
\mathsf{md}(\bot) & = & 0 & \mathsf{width}(\bot) & = & 0 \\
\mathsf{md}(p) & = & 0 & \mathsf{width}(p) & = & 0 \\
\mathsf{md}(\neg\psi) & = & \mathsf{md}(\psi) & \mathsf{width}(\neg\psi) & = & \mathsf{width}(\psi) \\
\mathsf{md}(\psi \wedge \theta) & = & \max\{\mathsf{md}(\psi), \mathsf{md}(\theta)\} & \mathsf{width}(\psi \wedge \theta) & = & \max\{\mathsf{width}(\psi), \mathsf{width}(\theta)\} \\
\mathsf{md}(\langle G \rangle^{\text{pre}}_{\geq k}\psi) & = & \mathsf{md}(\psi) + 1 & \mathsf{width}(\langle G \rangle^{\text{pre}}_{\geq k}\psi) & = & \max\{\mathsf{width}(\psi), k\} \\
\mathsf{md}(\langle P \rangle^{\text{suf}}_{\geq k}\psi) & = & \mathsf{md}(\psi) + 1 & \mathsf{width}(\langle P \rangle^{\text{suf}}_{\geq k}\psi) & = & \max\{\mathsf{width}(\psi), k\}
\end{array}
$$

Finally, we define some auxiliary formulae. Let φ be a formula of GPTL^{-} or $\mathrm{GPTL}^{-}_{\geq}$. We define

$$\langle G \rangle^{\text{pre}}_{=k}\varphi := \langle G \rangle^{\text{pre}}_{\geq k}\varphi \wedge \neg\langle G \rangle^{\text{pre}}_{\geq k+1}\varphi.$$

We also define $\langle P \rangle^{\text{suf}}_{=k}\varphi := \langle P \rangle^{\text{suf}}_{\geq k}\varphi \wedge \neg\langle P \rangle^{\text{suf}}_{\geq k+1}\varphi$.

Definition 12. The $\mathrm{GPTL}^{-}_{\geq}$**-type of depth** $d \in \mathbb{N}$ **and width** $k \in \mathbb{N}$ of a pointed word $(\mathcal{W}, i)$ where $\mathcal{W} = (V, E, \lambda)$, denoted by $\tau^{(\mathcal{W},i)}_{k,d}$, is defined recursively as follows. For $d = 0$, we define

$$\tau^{(\mathcal{W},i)}_{k,0} := \bigwedge_{\lambda(i)=p} p \wedge \bigwedge_{\lambda(i)\neq p} \neg p.$$

Assume we have defined $\tau^{(\mathcal{W},i)}_{k,d}$ for each pointed word $(\mathcal{W}, i)$ and let $T_{k,d}$ denote the set of all such types. We define $\tau^{(\mathcal{W},i)}_{k,d+1}$ intuitively as follows

$$
\begin{aligned}
\tau^{(\mathcal{W},i)}_{k,d+1} := \tau^{(\mathcal{W},i)}_{k,0} \wedge & \bigwedge_{\ell=0}^{k-1} \{\, \langle G \rangle^{\text{pre}}_{=\ell}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle G \rangle^{\text{pre}}_{=\ell}\tau \,\} \\
& \wedge \{\, \langle G \rangle^{\text{pre}}_{\geq k}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle G \rangle^{\text{pre}}_{\geq k}\tau \,\} \\
& \wedge \bigwedge_{\ell=0}^{k-1} \{\, \langle P \rangle^{\text{suf}}_{=\ell}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle P \rangle^{\text{suf}}_{=\ell}\tau \,\} \\
& \wedge \{\, \langle P \rangle^{\text{suf}}_{\geq k}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle P \rangle^{\text{suf}}_{\geq k}\tau \,\}.
\end{aligned}
$$

To ensure the uniqueness of each type, we use some canonical bracketing for them.

Lemma 13. *If two pointed words satisfy the same* $\mathrm{GPTL}^{-}_{\geq}$*-type of depth* d *and width* k*, then they cannot be separated by any formula of* $\mathrm{GPTL}^{-}_{\geq}$ *of depth* $\leq d$ *and width* $\leq k$*.*

Proof. We prove the claim by induction over d. Let k be fixed for the entire proof, and let $d = 0$. Pointed words $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfying the same $\mathrm{GPTL}^{-}_{\geq}$-type of depth 0 means that they agree on all tokens, and thus on all Boolean combinations thereof. Therefore they cannot be separated by any formula of depth ≤ 0, no matter its width.

Suppose then that the claim holds for all $d' < d$ and that $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy the same $\mathrm{GPTL}^{-}_{\geq}$-type of depth d and width k. We prove that $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ are indistinguishable by structural induction. Since $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy also the same $\mathrm{GPTL}^{-}_{\geq}$-types of depths $< d$ and width k, they are indistinguishable by any formula of depth $< d$ and width k. Thus it is sufficient to consider only the cases where $\langle G \rangle^{\text{pre}}_{\geq \ell}$ or $\langle P \rangle^{\text{suf}}_{\geq \ell}$ are applied for any $\ell \leq k$.

Suppose $\varphi = \langle G \rangle^{\text{pre}}_{\geq \ell}\psi$. Since ψ has depth $d - 1$, then by the induction hypothesis, the truth value of ψ at any vertex is determined by the truth value of the $\mathrm{GPTL}^{-}_{\geq}$-type of depth $d - 1$ and width k at that vertex. Consider

the set of all $(k, d-1)-$types τ that imply ψ:

$$S_\psi := \{\tau \in T_{k,d-1} \mid \tau \models \psi\}.$$

We write $n^{\text{pre}}_\tau(\mathcal{W}, i)$ for the number of vertices in the prefix whose $(k, d-1)$-type is τ. Note that $\mathcal{W}, i \models \langle G\rangle^{\text{pre}}_{\geq k}\psi$ if and only if $\sum_{\tau\in S_\psi} n^{\text{pre}}_\tau(\mathcal{W}, i) \geq k$. For equivalence, it thus suffices to show that $\sum_{\tau\in S_\psi} n^{\text{pre}}_\tau(\mathcal{W}, i) \geq k$ if and only if $\sum_{\tau\in S_\psi} n^{\text{pre}}_\tau(\mathcal{V}, i) \geq k$.

First, note that since the type $\tau^{(\mathcal{W},i)}_{k,d}$ records $\langle G\rangle^{\text{pre}}_{=\ell}$ for each $\ell \in \{0, \ldots, k-1\}$ and $\langle G\rangle^{\text{pre}}_{\geq k}$, it determines $\min(n^{\text{pre}}_\tau(\mathcal{W}, i), k)$ for each $\tau \in T_{k,d-1}$. Since $(\mathcal{V}, j)$ also satisfies $\tau^{(\mathcal{W},i)}_{k,d}$, this is equal to $\min(n^{\text{pre}}_\tau(\mathcal{V}, j), k)$. Consider now two cases:

1. If $n^{\text{pre}}_{\tau_0}(\mathcal{W}, i) \geq k$ for some $\tau_0 \in S_\psi$, then $\min(n^{\text{pre}}_{\tau_0}(\mathcal{W}, i), k) = \min(n^{\text{pre}}_{\tau_0}(\mathcal{V}, j), k) = k$, and hence both sums are at least k.
2. If $n^{\text{pre}}_\tau(\mathcal{W}, i) < k$ for all $\tau \in S_\psi$, then since $\min(n^{\text{pre}}_\tau(\mathcal{W}, i), k) = \min(n^{\text{pre}}_\tau(\mathcal{V}, j), k)$, we must have $n^{\text{pre}}_\tau(\mathcal{W}, i) = n^{\text{pre}}_\tau(\mathcal{V}, j)$, meaning the sums are equal.

We have thus shown that $\mathcal{W}, i \models \langle G\rangle^{\text{pre}}_{\geq\ell}\psi$ if and only if $\mathcal{V}, j \models \langle G\rangle^{\text{pre}}_{\geq\ell}\psi$. The case $\varphi = \langle P\rangle^{\text{suf}}_{\geq\ell}$ is handled analogously. □

Next, we define type automata for the above defined types. These are, intuitively, CPG-automata that have maximal ability to distinguish vertices, i.e., if two vertices have the same state after t iterations of a type automaton, they also have the same state after t iterations of any other CPG-automaton (that have the same width bound k for multisets as the type automaton). This notion is analogous to counting type automata in Ahvonen et al. (2024) and type automata in Kuusisto (2013).

A $\mathrm{GPTL}^-_{\geq}$**-type automaton of width** k is a CPG-automaton $A = (Q, \pi, \delta, n, b)$ defined as follows. We let Q be the set of all $\mathrm{GPTL}^-_{\geq}$-types of width k (up to depth n).[7] For the initializing function, we define

$$\pi(p) := p \wedge \bigwedge_{p\neq p'\in\Sigma} \neg p'.$$

For the transition function δ, let τ be a $\mathrm{GPTL}^-_{\geq}$-type of width k, and let M and M' be multisets of such types of the same depth as τ (when the depths of all types in $\{\tau\}, M$ and M' do not match, we may define transitions arbitrarily). We define

$$\begin{aligned}\delta(\tau, M, M') := \tau_0 \wedge &\bigwedge_{\ell=0}^{k-1} \{\, \langle G\rangle^{\text{pre}}_{=\ell}\sigma \mid M(\sigma) = \ell \,\} \\ &\wedge \{\, \langle G\rangle^{\text{pre}}_{\geq k}\sigma \mid M(\sigma) = k \,\} \\ &\wedge \bigwedge_{\ell=0}^{k-1} \{\, \langle P\rangle^{\text{suf}}_{=\ell}\sigma \mid M'(\sigma) = \ell \,\} \\ &\wedge \{\, \langle P\rangle^{\text{suf}}_{\geq k}\sigma \mid M'(\sigma) = k \,\},\end{aligned}$$

where τ_0 is the unique $\mathrm{GPTL}^-_{\geq}$-type of depth 0 that is consistent with τ.

Lemma 14. *Each formula φ of $\mathrm{GPTL}^-_{\geq}$ of modal depth d and width k has a logically equivalent finite disjunction of $\mathrm{GPTL}^-_{\geq}$-types of depth d and width k.*

Proof. Intuitively, we take the disjunction of all types of depth d and width k that imply φ and show that it is logically equivalent to φ.

[7] Here we abuse notation since formally Q should be a set of bit strings. This is not a problem since the set of all $\mathrm{GPTL}^-_{\geq}$-types of width k up to depth n is finite, and it can thus be mapped injectively to a set of bit strings of some length $m \in \mathbb{Z}_+$.

Let $T_{k,d}$ denote the set of all $\mathrm{GPTL}^-_{\geq}$-types of depth d and width k. Let $\Phi := \{\tau \in T_{k,d} \mid \tau \models \varphi\}$ and $\neg\Phi := \{\tau \in T_{k,d} \mid \tau \models \neg\varphi\}$, and let $\bigvee \Phi$ denote the disjunction of the types in Φ. Obviously, we have $\Phi \cap \neg\Phi = \emptyset$ and $\bigvee \Phi \models \varphi$. Note also that Φ is finite since $T_{k,d}$ is finite.

To show that $\varphi \models \bigvee \Phi$, it suffices to show that $\Phi \cup \neg\Phi = T_{k,d}$. Suppose that $\tau \in T_{k,d} \setminus (\Phi \cup \neg\Phi)$. Then there exist pointed words $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ that satisfy τ such that $\mathcal{W}, i \models \varphi$ and $\mathcal{V}, j \models \neg\varphi$. Since $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy the same $\mathrm{GPTL}^-_{\geq}$-type of depth d and width k, then by Lemma 13, there can be no formula of $\mathrm{GPTL}^-_{\geq}$ of modal depth d and width k that separates $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$, but φ is such a formula, which is a contradiction. Thus $\bigvee \Phi$ and φ are logically equivalent. □

Proposition 15. *If two pointed words $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy exactly the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth $d+1$, then $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy exactly the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth d.*

Proof. The proof is a straightforward induction over d. It is easy to see from the definition of $\mathrm{GPTL}^-_{\geq}$-types, that if $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy exactly the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth 1, then $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy exactly the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth 0.

Let d be such that the statement of the proposition holds. If $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy exactly the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth $d+1$, then the different parts of their $\mathrm{GPTL}^-_{\geq}$-type must be equal, i.e. the sets

$$\tau^{(\mathcal{W},i)}_{k,0},$$

$$\bigwedge_{\ell=0}^{k-1} \{\, \langle G\rangle^{\text{pre}}_{=\ell}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle G\rangle^{\text{pre}}_{=\ell}\tau \,\},$$

$$\{\, \langle G\rangle^{\text{pre}}_{\geq k}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle G\rangle^{\text{pre}}_{\geq k}\tau \,\},$$

$$\bigwedge_{\ell=0}^{k-1} \{\, \langle P\rangle^{\text{suf}}_{=\ell}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle P\rangle^{\text{suf}}_{=\ell}\tau \,\},$$

$$\{\, \langle P\rangle^{\text{suf}}_{\geq k}\tau \mid \tau \in T_{k,d} \text{ and } \mathcal{W}, i \models \langle P\rangle^{\text{suf}}_{\geq k}\tau \,\}$$

must be equal to their counterpart in $(\mathcal{V}, j)$ (when $(\mathcal{W}, i)$ is replaced with $(\mathcal{V}, j)$). By the induction hypothesis, the above observation also holds for $\tau \in T_{k,d-1}$. Thus $\tau^{(\mathcal{W},i)}_{k,d} = \tau^{(\mathcal{V},j)}_{k,d}$. □

Lemma 16. *Two pointed words $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy exactly the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth d if and only if they share the same state in each round up to d for each* CPG*-automaton.*

Proof. We will proceed by induction over the depth d. From left to right, we observe that the inputs to the automaton correspond in a natural way to the definition of types. From right to left, we observe that the result must hold for type automata where the transition function is injective.

Let us fix k, let $d = 0$, and let $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ be pointed words. $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth 0 if and only if $\tau^{(\mathcal{W},i)}_{k,0} = \tau^{(\mathcal{V},j)}_{k,0}$. From the definition of CPG-automata, $\tau^{(\mathcal{W},i)}_{k,0} = \tau^{(\mathcal{V},j)}_{k,0}$ holds if and only if each initialization function π assigns $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ the same initial state.

Let d be such that the statement of the lemma holds. From the definition of types, it is easy to see that two pointed words $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy exactly the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth $d+1$ if and only if:

1. They satisfy the same $\mathrm{GPTL}^-_{\geq}$-type of width k and modal depth 0.
2. For all $\mathrm{GPTL}^-_{\geq}$-types τ of width k and modal depth d, there are $l < k$ vertices in the prefix of $(\mathcal{W}, i)$ that satisfy τ if and only if there are l vertices in the prefix of $(\mathcal{V}, j)$ that satisfy τ.
3. For all $\mathrm{GPTL}^-_{\geq}$-types τ of width k and modal depth d, there are $l \geq k$ vertices in the prefix of $(\mathcal{W}, i)$ that satisfy τ if and only if there are at least k vertices in the prefix of $(\mathcal{V}, j)$ that satisfy τ.

4. For all $\mathrm{GPTL}^{-}_{\geq}$-types τ of width k and modal depth d, there are $l < k$ vertices in the suffix of $(\mathcal{W}, i)$ before i that satisfy τ if and only if there are l vertices in the suffix of $(\mathcal{V}, j)$ before j that satisfy τ.
5. For all $\mathrm{GPTL}^{-}_{\geq}$-types τ of width k and modal depth d, there are $l \geq k$ vertices in the suffix of $(\mathcal{W}, i)$ before i that satisfy τ if and only if there are at least k vertices in the suffix of $(\mathcal{V}, j)$ before j that satisfy τ.

Item 1. holds if and only if each initialization function π assigns $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ the same initial state. Notice that by Proposition 15, $\tau^{(\mathcal{W},i)}_{k,d} = \tau^{(\mathcal{V},j)}_{k,d}$.

Item 2. is equivalent to, there are $l < k$ vertices in the prefix of $(\mathcal{W}, i)$ with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d if and only if there are l vertices in the prefix of $(\mathcal{V}, j)$ with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d.

Item 3. is equivalent to, there are at least k vertices in the prefix of $(\mathcal{W}, i)$ with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d if and only if there are at least k vertices in the prefix of $(\mathcal{V}, j)$ with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d.

Item 4. is equivalent to, there are $l < k$ vertices in the suffix of $(\mathcal{W}, i)$ before i with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d if and only if there are l vertices in the suffix of $(\mathcal{V}, j)$ before j with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d.

Item 5. is equivalent to, there are at least k vertices in the suffix of $(\mathcal{W}, i)$ before i with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d if and only if there are at least k vertices in the suffix of $(\mathcal{V}, j)$ before j with the same $\mathrm{GPTL}^{-}_{\geq}$-type, τ, of width k and modal depth d.

From the induction hypothesis, if two pointed words $(\mathcal{W}, i')$ and $(\mathcal{V}, j')$ satisfy exactly the same $\mathrm{GPTL}^{-}_{\geq}$-type of width k and modal depth d, then they share the same state in each round up to d for each CPG-automaton. Thus from items 1-5, $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ share the same state in each round up to d for each CPG-automaton. Finally, from the transition function, δ, for each CPG-automaton, the state of $(\mathcal{W}, i)$ at round $d+1$ is determined by the state at round d and the respective multisets, $\mathcal{M}_k(Q)$, at round d. By the induction hypothesis and items 1-5, these multisets are the same for $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$. Thus, they share the same state in round $d+1$ for each CPG-automaton.

For the other direction, suppose that $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ are such that they share the same state in each round up to $d+1$ for each CPG-automaton. In particular, this holds for each type automaton. From Proposition 15, the transition function of a type automaton is injective (i.e. if two pointed words $(\mathcal{W}, i')$ and $(\mathcal{V}, j')$ satisfy different $\mathrm{GPTL}^{-}_{\geq}$-types of width k and modal depth d, then $(\mathcal{W}, i')$ and $(\mathcal{V}, j')$ satisfy different $\mathrm{GPTL}^{-}_{\geq}$-types of width k and modal depth $d+1$). Thus, that $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ share the same state in each round up to $d+1$ in a type automaton, implies that they have the same $\mathrm{GPTL}^{-}_{\geq}$-types of width k and modal depth $d+1$. □

We are now at last ready to prove Theorem 11.

Theorem 11 (Automata ⇒ logic)**.** *For each* CPG*-automaton, we can construct an equivalent tuple of formulae of* GPTL^{-}.

Proof. Intuitively, we construct a corresponding formula φ_q for each state q of the automaton such that φ_q is true in a vertex if and only if the final state given to the vertex is q. Then we construct a tuple consisting of suitable disjunctions of such formulae.

Let $A = (Q, \pi, \delta, n, b)$ be a CPG-automaton with multiset bound k. For each $q \in Q$, define

$$\Phi_q := \{\tau^{(\mathcal{W},i)}_{k,n} \in T_{k,d} \mid \text{ the state given by } A \text{ to } (\mathcal{W}, i) \text{ in round } n \text{ is } q\}.$$

We show that $\mathcal{W}, i \models \bigvee_{\tau \in \Phi_q} \tau$ if and only if the state given by A to $(\mathcal{W}, i)$ is q. If $\mathcal{W}, i \models \bigvee_{\tau \in \Phi_q} \tau$, then $\mathcal{W}, i \models \tau^{(\mathcal{V},j)}_{k,n}$ for some pointed Σ-labeled word $(\mathcal{V}, j)$ that outputs q. This means that $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ satisfy the same $\mathrm{GPTL}^{-}_{\geq}$-type of depth n and width k. By Lemma 16, this means that $(\mathcal{W}, i)$ and $(\mathcal{V}, j)$ share the same

state in A in each round up to n. Since A gives $(\mathcal{V}, j)$ the state q in round n, A also gives $(\mathcal{W}, i)$ the same state in round n. Conversely, if A gives $(\mathcal{W}, i)$ the state q in round n, then $\tau_{k,n}^{(\mathcal{W},i)} \in \Phi_q$ and thus $\mathcal{W}, i \models \bigvee_{\tau \in \Phi_q} \tau$.

Now we have a formula $\varphi_q = \bigvee_{\tau \in \Phi_q} \tau$ that is true in $(\mathcal{W}, i)$ exactly when i is in the state q in round n. For each $m \in [b]$, let Q_m denote the set of all states in Q where the mth bit is 1. Now we can construct the formula $\psi_m = \bigvee_{q \in Q_m} \varphi_q$, which is clearly true in $(\mathcal{W}, i)$ if and only if the mth output bit of $(\mathcal{W}, i)$ is 1. The equivalent formula tuple we seek is thus $(\psi_1, \ldots, \psi_b)$. □

We finish this section off by proving Theorem 3.

Theorem 3. *Encoder–decoder transformers without the final* $\mathrm{softmax}$*,* GPTL^- *and* CPG*-automata have the same expressive power.*

Proof. Theorems 9, 10 and 11 provide three of the translations. Translations from encoder–decoder transformers to GPTL^- and from CPG-automata to transformers are obtained by applying the translations back-to-back. From GPTL^- to CPG-automata, we first translate a tuple of GPTL^- into a transformer using Theorem 9. Then we modify the construction from the proof of Theorem 10 by increasing the output round by one and having the transition function essentially map the substrings representing the floating-point numbers 0 and 1 to the bits 0 and 1. □

D Proofs for Theorem 6

In this section, we prove Theorem 6.

Theorem 6. *Sentences of CPL, accepting encoder–decoder transformers and k-capped multiset functions recognize exactly the commutative star-free languages.*

We begin with an auxiliary result that has importance concerning the computability of some of the translations. An infinite sequence of tokens $\vec{\sigma} = \sigma_1 \sigma_2 \cdots$ is **ultimately periodic** if there are $p \geq 0$ and $n \geq 1$ such that $\sigma_{t+n} = \sigma_t$ for all $t > p$. For the least such values of p and n, we call $\sigma_1 \cdots \sigma_p$ the **preperiod** and $\sigma_{p+1} \cdots \sigma_{p+n}$ the **period** of $\vec{\sigma}$. A period is called **repetition-free** if it does not contain the same token twice. We say that a token-generating system is ultimately periodic if every output sequence it generates is either ultimately periodic or finite.

We first prove Proposition 7.

Proposition 7. *Each encoder–decoder transformer is ultimately periodic and the periods are repetition-free.*

Proof. By Theorem 5, we can turn the transformer into a sequence $\vec{\varphi}$ of formulae of GPTL^- that generates the same output sequences as the transformer with the same inputs. If the generated output with an input is finite, then we are done, so suppose that it is infinite. Intuitively, the proof rests on the observation that a formula φ of GPTL^- does not separate two vertices i and j in the suffix of a two-sorted word $\mathcal{W} = (V, E, \lambda)$ if **(1)** i and j share the same token, i.e., $\lambda(i) = \lambda(j)$, and **(2)** for each subformula of φ of the type $\langle P \rangle^{\mathrm{suf}} \psi$, we have $\mathcal{W}, i \models \langle P \rangle^{\mathrm{suf}} \psi$ if and only if $\mathcal{W}, j \models \langle P \rangle^{\mathrm{suf}} \psi$.

In more detail, let Φ be the (finite) set of subformulae of $\vec{\varphi}$, let i_t denote the last suffix vertex at generation step t, and define

$$H_t := \{\psi \in \Phi \mid \mathcal{W}, j \models \psi \text{ for some suffix vertex } j < i_t\}.$$

Now, the truth value of each $\psi \in \Phi$ at i_t at different generation steps t depends only on the pair $(\lambda(i_t), H_t)$ because the truth value of each subformula of the type $\langle G \rangle_{\geq k}^{\mathrm{pre}} \psi$ remains constant throughout the generation. Since the transformer is deterministic, both the next token $\lambda(i_{t+1})$ and $H_{t+1} = H_t \cup \{\psi \in \Phi \mid \mathcal{W}, i_t \models \psi\}$ also depend only on the pair $(\lambda(i_t), H_t)$. The sequence $(\lambda(i_t), H_t)_{t \geq 1}$ is thus obtained by iterating a function of the type $(\Sigma \cup \{\mathrm{BOS}\}) \times \mathcal{P}(\Phi) \to (\Sigma \cup \{\mathrm{BOS}\}) \times \mathcal{P}(\Phi)$, meaning it is ultimately periodic. Finally, since $(H_t)_{t \geq 1}$ is non-decreasing, H_t must be constant in the period of $(\lambda(i_t), H_t)_{t \geq 1}$. So, in the period, the only remaining varying state is the current token $\lambda(i_t)$, meaning it can only appear once per period of $(\lambda(i_t), H_t)_{t \geq 1}$, and thus also in the period of $(\lambda(i_t))_{t \geq 1}$. □

Next, we give the translation from sentences of CPL to accepting encoder–decoder transformers.

Lemma 17. *If a language is recognized by a sentence φ of counting propositional logic, then it is also recognized by a computable accepting encoder-decoder transformer.*

Proof. Intuitively, we modify and extend the transformer constructed in the proof of Theorem 9 to be autoregressive.

First, we translate φ into a transformer without the final softmax using Theorem 9. In a single additional decoder layer we use another MLP to calculate the truth value of $\neg\varphi$ as done for other subformulae in the construction in the proof of Theorem 9. According to the construction, the final linear transformation maps to a float vector of dimension 1 that has 1 if φ is true and 0 otherwise; we modify the linear transformation to map to float vectors of dimension $|\Sigma \cup \{\mathrm{EOS}\}|$ such that the truth value of φ is simply mapped to the accepting token and the truth value of $\neg\varphi$ is mapped to some non-accepting token, and other elements of the vector are zero.

The final softmax thus outputs a probability distribution where the corresponding token has the highest probability, and defining the token-generating function $f : \{0,1\}^n \to \Sigma \cup \{\mathrm{EOS}\}$ such that it simply picks the token with the highest probability gives us a transformer that immediately outputs an accepting token if φ accepts the input. Furthermore, since the transformer only speaks about the prefix input (because φ is a sentence of CPL), it generates the same token each round, and hence it never generates an accepting token if φ does not accept the input.

This translation is computable since the translation in Theorem 9 is computable and the modifications to it are obviously computable. □

Next, we translate accepting encoder–decoder transformers to bounded multiset functions.

Lemma 18. *If a language is recognized by an accepting encoder–decoder transformer, then it is also recognized by a computable k-capped multiset function for some $k \in \mathbb{N}$.*

Proof. Intuitively, by Theorem 5 we can consider CPG-automata instead of transformers and extract the result by looking at the state transition function.

Let T be an accepting transformer. By Theorem 5, we can turn T into an equivalent autoregressive CPG-automaton A with some multiset bound k. On the input side (i.e. the prefix), the equivalence classes induced by the tokens can never be refined further, meaning that if two input vertices share the same token, they also share the same state in every round. This is because the first input of the transition function is the token corresponding to the vertex and the other two inputs are shared across encoder vertices. Therefore, the maximal information ever obtained regarding the input is the k-capped multiset of tokens appearing in the input. This implies that any word generated by A (and thus acceptance by T) is invariant under (i) permuting the input and (ii) inserting a copy of a token σ that already occurs at least k times in the input. This means the language recognized by T can also be recognized by a k-capped multiset function.

Finding out which k-capped multisets should map to 1 is simple. We simply run the transformer on each input where each token appears at most k times; for any input where a token appears more than k times, the output sequence would be the same if the token only appeared k times, so they do not need to be considered. By Proposition 7, the transformer only needs to run for a bounded number of iterations before its output sequence starts repeating itself, at which point we can terminate if an accepting token has not been generated. Thus, the translation above is computable. □

We close the loop by translating bounded multiset functions to sentences of CPL.

Lemma 19. *If a language is recognized by a k-capped multiset function for some $k \in \mathbb{N}$, then it is also recognized by a computable sentence of counting propositional logic.*

Proof. We essentially construct a canonical formula based on the k-capped multisets that are mapped to 1.

Each k-capped multiset M can be expressed as a conjunction of formulas $\langle G \rangle^{\mathrm{pre}}_{=\ell}\sigma$ for each token σ such that $M(\sigma) = \ell < k$ and $\langle G \rangle^{\mathrm{pre}}_{\geq k}\sigma$ for each σ such that $M(p) = k$. We take the disjunction of the formulae corre-

sponding to k-capped multisets that the function maps to 1. Clearly the resulting sentence is true in a word if and only if the corresponding multiset is accepted by the function.

This translation is obviously computable. □

We complete the proof by showing that k-capped multiset functions recognize exactly the commutative star-free languages. A language L over Σ is **commutative** if it is closed under permutation, i.e., if $\vec{\sigma} \in L$ implies that every permutation of $\vec{\sigma}$ is in L. For languages L, L' over Σ, their **concatenation** is $LL' := \{ \vec{\sigma}\vec{\sigma}' \mid \vec{\sigma} \in L,\ \vec{\sigma}' \in L' \}$ and the **complement** of L is $\overline{L} := \Sigma^* \setminus L$. The class of **star-free** languages over Σ is the smallest class of languages that contains $\emptyset$ and $\{\sigma\}$ for each $\sigma \in \Sigma$ and is closed under union, concatenation and complementation.

Lemma 20. *A language L is recognized by a k-capped multiset function for some $k \in \mathbb{N}$ if and only if L is commutative and star-free.*

Proof. From left to right, we intuitively construct a regular expression for the language L. For the converse we utilize Schützenberger's theorem (Schützenberger, 1965) which states that star-free languages are counter-free, which essentially means that, in terms of membership, the number of occurrences of a token only matters up to some bound.

($\Rightarrow$): Let L be a language recognized by some k-capped multiset function $\mathcal{M}_k(\Sigma) \to \{0, 1\}$. We construct a regular expression that defines L as follows. First, given a multiset $M \in \mathcal{M}_k(\Sigma)$, consider the set C consisting of those words that contain exactly $M(p)$ occurrences of each token p. Let $\mathcal{C}$ be the union of such sets C across all multisets $M \in \mathcal{M}_k(\Sigma)$. We call each $w \in \mathcal{C}$ a canonical input, because the question of whether an input is accepted or not by the function can be reduced to asking whether a corresponding input in $\mathcal{C}$ is accepted.

Next, let $w = \sigma_1 \cdots \sigma_n \in \mathcal{C}$ be a canonical input and let $S \subseteq \Sigma$ be the set of tokens σ for which σ is saturated in w, i.e., for which σ appears k times in w (the maximum number of occurrences possible). Now S is a language easily definable as the union of languages consisting of single tokens. We define the language S^* consisting of those words that only contain saturated tokens via the regular expression

$$S^* := \overline{\Sigma^*(\Sigma \setminus S)\Sigma^*},$$

noting that Σ^* is expressible without the Kleene star as $\overline{\emptyset}$.

Now the language $\{w\}$ is definable as the concatenation of the languages $\{\sigma_1\}, \ldots, \{\sigma_n\}$. We can insert S^* into $n + 1$ positions between any of the letters $\sigma_1, \ldots, \sigma_n$, including at the very front before σ_1 and at the very back after σ_n. There are 2^{n+1} possible ways to insert S^* into zero or more such positions, each yielding a language. We take the union of these languages, which contains exactly the words obtained from w by adding saturated tokens. We obtain such a language for each canonical input w whose corresponding multiset is classified 1 by the function, and the union of these languages is exactly L, the language recognized by the function.

($\Leftarrow$): Let $L \subseteq \Sigma^*$ be commutative and star-free. We will show that there exists some $n \in \mathbb{N}$ such that for all words $w, w' \in \Sigma^*$ that share the same n-capped multiset of tokens, we have $w \in L \iff w' \in L$. This implies that L is recognizable by a function $\mathcal{M}_n(\Sigma) \to \{0, 1\}$.

By Schützenberger's theorem (Schützenberger, 1965), used in the form presented by McNaughton & Papert (1971), since L is star-free, it is **counter-free**, i.e., there is an $n \in \mathbb{N}$ such that

$$xy^m z \in L \iff xy^n z \in L$$

for all $x, y, z \in \Sigma^*$ and $m \geq n$. Now let w and w' be words with the same n-capped multiset. First, by commutativity, permute w or w' so that all identical tokens are next to each other and occur in the same order in both w and w'. Then, a token σ is either present exactly the same number of times in both words, so no adjustment is needed, or both counts are at least n, so the counter-free property of L lets us add however many repetitions of the token to the word in which it appears less as is needed. Repeating this for every token σ turns the words identical without affecting membership in L, so $w \in L \iff w' \in L$. □

We are now ready to prove Theorem 6.

Theorem 6. *Sentences of CPL, accepting encoder–decoder transformers and k-capped multiset functions recognize exactly the commutative star-free languages.*

Proof. Lemmas 17, 18 and 19 together imply that all three classes recognize exactly the same languages. By Lemma 20, these are the commutative star-free languages. □

E PROOFS FOR THEOREM 8

In this section, we prove Theorem 8.

Theorem 8. *As token-generating systems, autoregressive encoder–decoder transformers and sentences of CPL (without autoregression) are equally expressive regardless of if the transformers use the final* $\mathrm{softmax}$ *or not.*

We start with the direction from transformers to logic.

Lemma 21. *For each autoregressive encoder–decoder transformer, we can construct a generation-equivalent pair of matrices of sentences of CPL.*

Proof. We combine three facts: **(1)** a transformer can only distinguish a finite number of inputs, **(2)** autoregressive transformers are ultimately periodic, and **(3)** a sentence of counting propositional logic can distinguish the same inputs as a transformer. Intuitively, **(1)** means that we only have to study a finite number of inputs, **(2)** guarantees that every output starts repeating after a bounded number of iterations, and **(3)** means that we can separate the tokens generated by the transformer in each position up to the point where they inevitably start repeating via sentences of CPL.

Let T be an autoregressive transformer and let $f : \{0,1\}^n \to \Sigma \cup \{\mathrm{EOS}\}$ be its token-generating function. Observe that there is a $k \in \mathbb{Z}_+$ such that the word generated by T depends only on the k-capped multiset of tokens present in the input (see Lemma 18). Since there are only finitely many such multisets, we may fix a finite set I of inputs containing one representative per multiset. Let $\#_\sigma(\mathcal{W}_p)$ denote the number of σ-tokens in the input $\mathcal{W}_p$. For each $\mathcal{W}_p \in I$, define a sentence $\chi_{\mathcal{W}_p}$ of counting propositional logic stating that an input has the same k-capped multiset as $\mathcal{W}_p$ as follows:

$$\chi_{\mathcal{W}_p} := \bigwedge_{\#_\sigma(\mathcal{W}_p)<k} \langle G \rangle^{\mathrm{pre}}_{=\#_\sigma(\mathcal{W}_p)} \sigma \;\wedge \bigwedge_{\#_\sigma(\mathcal{W}_p)\geq k} \langle G \rangle^{\mathrm{pre}}_{\geq k} \sigma.$$

Since Σ is finite, $\chi_{\mathcal{W}_p}$ is a finitary formula.

For each $\mathcal{W}_p \in I$, let $\vec{\sigma}^{\mathcal{W}_p}$ be the token sequence generated by T on the input $\mathcal{W}_p$, padded into an infinite sequence with EOS-tokens if it is finite. By Proposition 7, each $\vec{\sigma}^{\mathcal{W}_p}$ is ultimately periodic and the period is repetition-free; let $p_{\mathcal{W}_p}$ denote the length of its preperiod and $n_{\mathcal{W}_p}$ the length of its period, and set $N := \max\{\, p_{\mathcal{W}_p} \mid \mathcal{W}_p \in I \,\}$ and $L := \max\{\, n_{\mathcal{W}_p} \mid \mathcal{W}_p \in I \,\}$.

We are now ready to define our sentence matrices. For each word $\mathcal{W}_p \in I$, let $\mathcal{W}_s$ denote the suffix of length $N + L$ obtained by applying T to $\mathcal{W}_p$ autoregressively (padded with EOS-tokens if the generation is finite). Let $T(\mathcal{W}_s(i)) \in \{0,1\}^n$ denote the bit-string output of the transformer at the ith node of the suffix. For $1 \leq r \leq n$, $1 \leq m \leq N$ and $1 \leq \ell \leq L$, let the rth sentence of the mth column of Ψ_1 and of the ℓth column of Ψ_2 be

$$\bigvee\{\, \chi_{\mathcal{W}_p} \mid \mathcal{W}_p \in I, \text{ the } r\text{th bit of } T(\mathcal{W}_s(m)) \text{ is } 1 \,\} \text{ and } \bigvee\{\, \chi_{\mathcal{W}_p} \mid \mathcal{W}_p \in I, \text{ the } r\text{th bit of } T(\mathcal{W}_s(N+\ell)) \text{ is } 1 \,\},$$

respectively. Fix an input and let $\mathcal{W}_p \in I$ be its representative. Since it satisfies $\chi_{\mathcal{W}_p}$ and no other $\chi_{\mathcal{W}_p'}$, each column of the pair encodes the bit-string output of T on the corresponding node of the suffix, and thus, via f, the token that T generates: the preperiod columns encode the tokens of the first N steps, and the period columns encode the tokens of steps $N + 1$ through $N + L$.

The pair thus generates the same tokens as T on the first N steps. Now consider what T does after step N. Since N is at least the preperiod of $\vec{\sigma}^{\mathcal{W}_p}$, T is already inside its cycle, so from step $N + 1$ onward it simply repeats its period of length $n_{\mathcal{W}_p} \leq L$, whose tokens are pairwise distinct. The pair, meanwhile, outputs the encoded tokens of steps $N + 1, N + 2, \ldots$ in order until it is about to output a token that it has already output since the start of the period. The first $n_{\mathcal{W}_p}$ of these tokens are one full traversal of T's cycle, hence all distinct, so no reset happens among them; the next token, whether it is encoded by the $(n_{\mathcal{W}_p} + 1)$th period column or by the first one again if $n_{\mathcal{W}_p} = L$, is the first token of the cycle, which has already been output. So the pair resets exactly when T starts its cycle over, and both go on repeating the same $n_{\mathcal{W}_p}$ tokens forever. Therefore the transformer and the sentence matrices are generation-equivalent. □

We then show the direction from logic to transformers.

Lemma 22. *For each pair of matrices of sentences of CPL, we can construct a generation-equivalent autoregressive encoder–decoder transformer that uses the final* $\mathrm{softmax}$.

Proof. We intuitively construct the transformer such that it first simulates the columns of the first matrix from left to right, keeping track of the current column via the masked self-attention in decoder layers. Then, the transformer simulates the columns of the second matrix from left to right, keeping track of the current column via the last generated token; if a token repeats, the transformer returns to the beginning of the second matrix.

Let Ψ_1, Ψ_2 be matrices of CPL-sentences with n rows, let N and L be their numbers of columns, and let $f : \{0,1\}^n \to \Sigma \cup \{\mathrm{EOS}\}$ be the token-generating function. Fix an input $\mathcal{W}$, let $\vec{\sigma}_1$ and $\vec{\sigma}_2$ be the token strings obtained from the binary matrices $\Psi_1(\mathcal{W})$ and $\Psi_2(\mathcal{W})$ via f, and let $\vec{\sigma}'_2$ be the longest prefix of $\vec{\sigma}_2$ that does not contain any token twice. Since the tokens $\sigma_1, \ldots, \sigma_k$ of $\vec{\sigma}'_2$ are pairwise distinct, the successor function $\mathrm{succ}(\sigma_i) := \sigma_{i+1}$ for $i < k$ and $\mathrm{succ}(\sigma_k) := \sigma_1$ is well-defined.

We construct a transformer T with output dimension $|\Sigma \cup \{\mathrm{EOS}\}|$ and a final $\mathrm{softmax}$, together with a token-generating function f', such that (T, f') is generation-equivalent to (Ψ_1, Ψ_2) with f. The transformer computes, at every suffix vertex, three pieces of information. First, the truth values of all formulae occurring in Ψ_1 and Ψ_2, via cross-attention to the decoder as in Theorem 9. Second, the current token, via the embedding. Third, the position t of the vertex in the suffix, capped at $N+1$, via $N+1$ decoder layers that together check the number of preceding suffix vertices, i.e., they compute the truth values of the formulas $\langle P \rangle^{\mathrm{suf}} \top$, $\langle P \rangle^{\mathrm{suf}} \langle P \rangle^{\mathrm{suf}} \top$, etc.

Since the domain of f is finite, an MLP can then use this information to implement the following look-up table to ensure that the correct token σ is generated at position t: if $t \leq N$, then σ is the token encoded by the tth column of $\Psi_1(\mathcal{W})$; if $t = N+1$, then σ is the token encoded by the first column of $\Psi_2(\mathcal{W})$; if $t \geq N+2$, then $\sigma := \mathrm{succ}(\sigma')$ where σ' is the current token. Finally, the linear transformation places a logit C on the coordinate of σ and 0 elsewhere, where C is a float large enough that $\exp_{\mathcal{F}}(-C) = 0$; after $\mathrm{softmax}$, the coordinate of σ has probability 1 and all others exactly 0. The token-generating function f' can then be defined to read a $\mathrm{softmax}$ output such that it returns the unique token with non-zero probability, so T generates σ. So at each step, (T, f') generates the same token as (Ψ_1, Ψ_2) with f, meaning they are generation-equivalent. □

We are now able to prove Theorem 8.

Theorem 8. *As token-generating systems, autoregressive encoder–decoder transformers and sentences of CPL (without autoregression) are equally expressive regardless of if the transformers use the final* $\mathrm{softmax}$ *or not.*

Proof. The direction from transformers to logic is shown by Lemma 21 and the opposite direction is shown by Lemma 22. □

F Variations of the transformer architecture

In this section, we show that allowing a cross-attention layer to have multiple heads or adding layer normalization does not increase expressive power, and that we can obtain analogous results for non-strict masking with minor modifications to the proofs.

F.1 Multi-head cross-attention

Many papers note in passing that a multi-head self-attention layer with h heads can be simulated with h sequential single-head self-attention layers (Merrill & Sabharwal, 2024; Li et al., 2024). Here, we prove the same for cross-attention.

Theorem 23. *A multi-head cross-attention layer with h heads can be simulated by h sequential single-head cross-attention layers.*

Proof. We intuitively have a dedicated space in the internal dimension for each attention head, and the heads are computed one-by-one instead of all at once.

Let $X \in \mathcal{F}^{n_x \times d}$ (the encoder output) and $Y \in \mathcal{F}^{n_y \times d}$ (the decoder input) be the inputs to a multi-head cross-attention layer with h heads. Recall that

$$\mathrm{MHCA}(X, Y) = \mathrm{concat}(H^{(1)}(X, Y), \ldots, H^{(h)}(X, Y))W^O,$$

where each head $H^{(\ell)}(X, Y) \in \mathcal{F}^{n_y \times d_v}$ is a single-head cross-attention computation with its own parameter matrices $W_\ell^Q, W_\ell^K, W_\ell^V$, and $W^O \in \mathcal{F}^{hd_v \times d}$ is the output projection.

We simulate this with h sequential single-head cross-attention layers. The key idea is to use h designated "slots" of width d_v in the hidden dimension of Y to accumulate the outputs of each head one at a time. We reserve dimensions $(\ell - 1)d_v + 1, \ldots, \ell d_v$ of Y for the ℓ-th head's output, assuming $d \geq hd_v$ (otherwise we widen Y and X by padding with zeros, which does not affect expressivity).

At layer ℓ, we apply a single-head cross-attention layer with parameters $(W_\ell^Q, W_\ell^K, W_\ell^V)$, producing $H^{(\ell)}(X, Y) \in \mathcal{F}^{n_y \times d_v}$. We then use the residual connection and an MLP to write this output into the ℓ-th slot of Y, leaving the other slots unchanged. Specifically, the MLP can be configured to zero out any interference from the residual connection in the target slot and to preserve the values in all other slots.

After h layers, the hidden state contains $H^{(1)}(X, Y), \ldots, H^{(h)}(X, Y)$ concatenated in the reserved slots, exactly as in the output of $\mathrm{concat}(H^{(1)}(X, Y), \ldots, H^{(h)}(X, Y))$. The final linear projection W^O can be implemented by the MLP of the last layer.

Since the encoder output X is a fixed side input and is not modified by any of the h layers, each single-head cross-attention layer sees the same X as the original multi-head layer, and the computation within each head is identical to the corresponding head in the multi-head layer. Therefore the output of the h-layer simulation equals $\mathrm{MHCA}(X, Y)$. □

For the rest of the Appendix, we thus assume that each self- or cross-attention layer contains only a single attention head, which makes the proofs simpler.

F.2 Masking

It has been shown that transformers with strict causal masking are strictly more expressive than those with non-strict causal masking (Yang et al., 2024; Li & Cotterell, 2025). We note that our characterizations easily cover both strict and non-strict causal masking with only slight modification to the logic and automaton class.

First, we introduce the alternative encoder–decoder architecture. It is otherwise the same as the one we have presented in Section 2.3, except that the diagonal is not masked in the decoder layers, i.e., we use the masking function

$$(\mathrm{mask}(S))_{ij} = \begin{cases} -\infty & \text{when } j > i, \\ S_{ij} & \text{otherwise.} \end{cases}$$

The difference to the main section is that equality is not included in the upper condition.

Next, consider the logic GPTL_2^- obtained from GPTL^- by replacing the modality $\langle P \rangle^{\mathrm{suf}}$ with the modality $\langle P \rangle_{\geq k}^{\mathrm{suf}}$, with the following semantics. Given a pointed two-sorted word $(\mathcal{W}, i)$, we have

$$\mathcal{W}, i \models \langle P \rangle_{\geq k}^{\mathrm{suf}} \varphi \text{ if and only if } |\{j \in V(\mathcal{W}_s) \mid j \leq i \text{ and } \mathcal{W}, j \models \varphi\}| \geq k.$$

Note the difference to the abbreviation defined in Appendix C.3, since here $\langle P \rangle_{\geq k}^{\mathrm{suf}}$ can potentially count the vertex i itself.

Thirdly, consider CPG'-automata where the automata are otherwise defined the same as CPG-automata in Section 2.5 except that the second multiset of states is received from prior vertices in the suffix *including the vertex itself*.

We obtain the following theorem.

Theorem 24. *Encoder–decoder transformers with non-strict causal masking, the logic* GPTL_2^- *and* CPG'*-automata have the same expressive power.*

Proof. The translations from transformers to automata and from automata to logic are analogous to the case with GPTL^- and CPG-automata. We only need to modify the translation from logic to transformers, where

we intuitively show that the decoder can simulate the modality $\langle P \rangle^{\text{suf}}_{\geq k}$ similarly to how the encoder and decoder layers can simulate the modality $\langle G \rangle^{\text{pre}}_{\geq k}$.

We modify the proof of Theorem 9 from Appendix C.1 by constructing a stack of decoder layers that simulates the modality $\langle P \rangle^{\text{suf}}_{\geq k}$ with the semantics defined above. The matrices W_Q and W_K are defined exactly as we did for the decoder layers that simulate $\langle G \rangle^{\text{pre}}_{\geq k}$. This means that before masking, each row contains the column of the input matrix that encodes the truth values of φ multiplied by F, the largest positive non-infinite float in the format. After masking, entries to the right of the diagonal are set to $-\infty$, which are then set to 0 by $\mathrm{softmax}$. For the other entries on each row, if at least one of them is F, then the bias and $\mathrm{softmax}$ turn each 0 into 0 and each F into $\frac{1}{\ell'}$ (rounded to the nearest float) where ℓ' is the sum of ℓ 1s and ℓ is the number of prior vertices (including self) that satisfy φ. This is the same outcome as in the proof of Theorem 47 of Ahvonen et al. (2026a), which means that we can construct the rest of the decoder in the same way, constructing the value matrix to trigger underflow when $\ell \geq k$.

Just like in Ahvonen et al. (2026a), there is the possibility of false positives; if all unmasked entries on a row are 0, then biasing and $\mathrm{softmax}$ replaces each 0 with $\frac{1}{m'}$ (rounded to the nearest float) where m' is the sum of m 1s and m is the number of prior vertices in the suffix including the vertex itself. Applying the value matrix construction then outputs 0 for these rows, because each row of XW_V corresponding to these non-zero entries is 0. This is the same result as with underflow, so from the point of view of the vertex, the cases $\ell = 0$ and $\ell \geq k$ appear indistinguishable. To rectify this, we can simply use the masked self-attention head constructed in the proof of Theorem 9 that simulates the non-counting modality $\langle P \rangle^{\text{suf}}$, which also works for the non-strict version of the modality and is enough to distinguish the cases $\ell = 0$ and $\ell \geq k$.

The rest of the proof does not require modifications. □

We note that we can obtain results analogous to Theorems 4, 5, 6 and 8 and Proposition 7 by modifying the proofs to consider non-strict masking instead of strict masking, GPTL^{-}_{2} instead of GPTL^{-} and CPG'-automata instead of CPG-automata. In particular, in the proof of Proposition 7 the set H_t still saturates even with the non-strict past modality.

Alongside non-strict causal masking, Vaswani et al. (2017) also shifts the decoder inputs to the right. However, this occurs outside the calculations of the transformer and thus does not affect our translations.

F.3 Layer normalization

It is known that layer normalization in a decoder self-attention layer does not affect expressive power (Chiang & Cholak, 2022; Li & Cotterell, 2025, Proposition B.13). In this subsection, we show that the same holds for cross-attention, and that the choice between pre-norm and post-norm does not affect expressivity.

For each $\mathbf{x} \in \mathcal{F}^*$, let

$$\mathrm{LN}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \cdot \gamma + \beta$$

be a **layer normalization** function, where $\mu \in \mathcal{F}$ and $\sigma \in \mathcal{F}$ are the mean and standard deviation of the elements in $\mathbf{x}$, $\gamma, \beta \in \mathcal{F}$ are learned parameters, and $\varepsilon \in \mathcal{F}$ is a small constant for numerical stability. A **decoder layer with residual connection and pre-norm** is obtained by replacing the multi-head cross-attention layer with residual connection ($\mathrm{MHCA}(X, Y) + Y$) with

$$\mathrm{MHCA}(\mathrm{LN}(X), \mathrm{LN}(Y)) + Y,$$

and one with **post-norm** is obtained by replacing it instead with

$$\mathrm{LN}(\mathrm{MHCA}(X, Y) + Y),$$

where LN is applied row-wise.

Theorem 25. *A decoder layer with layer normalization (pre-norm or post-norm) can be simulated by a stack of decoder layers without layer normalization.*

Proof. When the dimension of vectors is fixed to d, $\mathrm{LN} : \mathcal{F}^d \to \mathcal{F}^d$ is a function on a finite domain and can therefore be implemented as a look-up table by a sufficiently wide two-layer ReLU MLP. For post-norm, LN is applied after CA and before the MLP sublayer, so we can use the MLP to simulate LN and add a layer right after

that carries out the other calculations of the MLP. For pre-norm, LN is applied to the inputs before CA, so we add a layer right before where the MLP simulates LN. □

Theorem 26. *A decoder layer without layer normalization can be simulated by a decoder layer with layer normalization (pre-norm or post-norm).*

Proof. We adapt the *mirror dimension* trick of Chiang & Cholak (2022) to cross-attention, meaning that we double the dimension and mirror the values on the left with inverse values on the right. Given inputs $X \in \mathcal{F}^{n_x \times d}$ and $Y \in \mathcal{F}^{n_y \times d}$, define mirrored versions $\tilde{X} \in \mathcal{F}^{n_x \times 2d}$ and $\tilde{Y} \in \mathcal{F}^{n_y \times 2d}$ by setting $\tilde{X}_{i,d+j} = -X_{i,j}$ and similarly for $\tilde{Y}$. Every row of a mirrored matrix thus has mean $\mu = 0$.

We show that cross-attention preserves this structure. Choose query and key projections $\tilde{W}^Q = \binom{W^Q}{0}$ and $\tilde{W}^K = \binom{W^K}{0}$, so that the attention scores depend only on the original halves and are identical to those of the unmirrored computation. Choose the value projection $\tilde{W}^V = \left(\begin{smallmatrix} W^V & -W^V \\ 0 & 0 \end{smallmatrix}\right)$, so that $\tilde{X}\tilde{W}^V = (XW^V, -XW^V)$, which is mirrored. Since the attention weights multiply this from the left, the cross-attention output is mirrored, and its first d columns equal the original cross-attention output. The residual $+\tilde{Y}$ preserves mirroring since mirrored matrices are closed under addition.

Following Chiang & Cholak (2022), we control the row variance by reserving additional padding dimensions whose values are set (via the MLP sublayer) so that every row has the same norm. With $\mu = 0$ and σ^2 constant across all rows, choosing $\gamma = \sqrt{\sigma^2 + \varepsilon}$ and $\beta = 0$ makes LN the identity. The argument applies to both pre-norm and post-norm, since in either case LN is applied to a mirrored matrix with controlled norms. □