TRANSFORMERS ARE INHERENTLY SUCCINCT

Pascal Bergsträßer
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany
bergstraesser@cs.uni-kl.de

Ryan Cotterell
ETH Zürich
Zürich, Switzerland
ryan.cotterell@inf.ethz.ch

Anthony W. Lin
RPTU Kaiserslautern-Landau and MPI-SWS
Kaiserslautern, Germany
lin@cs.uni-kl.de

ABSTRACT

We propose succinctness as a measure of the expressive power of a transformer in describing a concept. To this end, we prove that transformers are highly expressive in that they can represent formal languages substantially more succinctly than standard representations of formal languages like finite automata and Linear Temporal Logic (LTL) formulas. As a by-product of this expressivity, we show that verifying properties of transformers is provably intractable (i.e. EXPSPACE-complete).

1 INTRODUCTION

Transformers (Vaswani et al., 2017) are the underlying model behind the recent success of Large Language Models (LLMs). The past few years saw a large amount of theoretical development explaining the expressive power of transformers (Strobl et al., 2024; Barceló et al., 2024; Yang et al., 2024; Hahn, 2020; Pérez et al., 2021; Chiang & Cholak, 2022; Jerad et al., 2025), their trainability and length generalizability (Zhou et al., 2024; Huang et al., 2025; Chiang & Cholak, 2022), and the extent to which one can formally verify them (Sälzer et al., 2025). Interestingly, it is known that transformers with fixed (finite) precision (Yang et al., 2024; Barceló et al., 2024; Jerad et al., 2025; Li & Cotterell, 2025) recognize a well-known subclass of regular languages called *star-free languages*. Fixed-precision transformers are especially pertinent to real-world transformers, which are implemented on hardware with fixed (finite) precision.

Star-free languages form a rather small subclass of regular languages. More precisely, a star-free regular expression allows the intersection and complementation operators instead of the Kleene star. For this reason, the regular language a^*b^* is star-free because it can be defined as $\overline{\overline{\emptyset}.b.a.\overline{\emptyset}}$. On the other hand, it is known that regular languages like $(aa)^*$ are not star-free (cf. see (Straubing, 1994)). This is in contrast to Recurrent Neural Networks (RNN), which can recognize all regular languages (Siegelmann & Sontag, 1995; Merrill et al., 2020). Thus, expressivity as language recognizers per se is perhaps not the most useful criterion for an LLM architecture.

In this paper, we propose *succinctness* as an alternative angle in understanding the "expressivity" of transformers. More precisely, the succinctness of a language L with respect to a class $\mathcal{C}$ of language recognizers (e.g. transformers, automata, etc.) measures the smallest (denotational) size of $T \in \mathcal{C}$ that recognizes L, i.e., how many symbols are used to describe T. Succinctness has been studied in logic in computer science (e.g. (Grohe & Schweikardt, 2004; Stockmeyer, 1974)) as an alternative (and more computational) measure of expressiveness, and has direct consequence in how computationally difficult it is to analyze a certain expression. For example, Linear Temporal Logic (LTL) (Pnueli, 1977) is expressively equivalent to star-free regular languages (e.g. see (Libkin, 2004)), as well as a subclass of deterministic finite automata called *counter-free automata* (McNaughton & Papert, 1971). Despite this, it is known that LTL can be exponentially more succinct than finite automata (Sistla & Clarke, 1985). In other words, certain concepts can be described considerably more succinctly by LTL formulas as by finite automata. This has various consequences, e.g., analyzing

LTL formulas (e.g. checking whether they describe a trivial concept) is provably computationally more difficult than analyzing finite automata (Sistla & Clarke, 1985).

Contributions. Our main result can be summarized as follows:

Transformers can describe concepts extremely succinctly.

More precisely, we show that transformers can be *exponentially* more succinct than LTL and RNN (so including state-of-the-art State-Space Models (SSMs), e.g., see (Gu & Dao, 2023; Merrill et al., 2024)), and *doubly exponentially* more succinct than finite automata. This means that, with the same descriptional size, transformers can encode complex patterns that require exponentially (resp. doubly exponentially) larger descriptional sizes for LTL and RNN (resp. automata). As a by-product of this expressivity, one may surmise that analyzing transformers must be computationally challenging. We show this to be the case. That is, verifying simple properties about transformers (e.g. whether it recognizes a trivial language) is computationally difficult: EXPSPACE-complete. That is, with standard complexity-theoretic assumptions, this cannot be done in better than *double exponential time*.

In fact, we also show a matching upper bound on the succinctness gap for LTL (exponential) and automata (double exponential). That is, we provide a translation from fixed-precision transformers to exponential-sized LTL formulas. This significantly improves the previously shown doubly exponential translation by Yang et al. (2024). As a consequence, for any fixed-precision transformer, there is an LTL formula (resp. finite automaton) of (doubly) exponential size recognizing the same language.

In proving our succinctness results, we show how transformers can count from 0 up to 2^{2^n}, i.e., the so-called "(doubly exponentially) large counters". This requires a subtle encoding of large counters using attention. We then prove that the resulting languages using LTL and RNN (resp. finite automata) require exponentially (resp. doubly exponentially) larger description.

What assumptions do we use in our results? We assume that transformers and RNN are of a *fixed (finite) precision*. This assumption is faithful to real-world implementations, which use only fixed-precision arithmetics. In this paper, we use *Unique-Hard Attention Transformers (UHATs)* model, which is a simple and popular abstraction of self-attention (cf. (Yang et al., 2024; Jerad et al., 2025; Strobl et al., 2024; Hao et al., 2022; Li & Cotterell, 2025; Hahn, 2020; Barceló et al., 2024; Bergsträßer et al., 2024)). In particular, recent works by Jerad et al. (2025) show that expressivity bounds on UHATs entail bounds on realistic softmax transformers with a fixed precision.

Organization. We recall some formal concepts (transformers, automata, and logic) in Section 2. We show in Section 3 how transformers can encode an exponential tiling problem. We show in Section 4 how this implies succinctness of UHATs relative to other representations. Applications of our results for reasoning about transformers are discussed in Section 5. We conclude the paper in Section 6.

2 PRELIMINARIES

We often denote vectors by boldface letters and for a vector $\boldsymbol{v} = (v_1, \ldots, v_d)$ we write $\boldsymbol{v}[i, j] := (v_i, \ldots, v_j)$ for all $1 \leq i \leq j \leq d$ and if $i = j$, we simply write $\boldsymbol{v}(i)$. We also write $\boldsymbol{n}$ for a number n to denote a vector $(n, \ldots, n)$ of appropriate dimension.

An *alphabet* is a finite set Σ of *symbols* (a.k.a. *tokens*). We write Σ^* for the set of all *words* (a.k.a. *sequences*, *strings*) of the form $a_1 \ldots a_n$, where $n \geq 0$ and $a_i \in \Sigma$ for all $i \in [1, n]$. We write Σ^+ for the set of all non-empty words. A *language* is a subset $L \subseteq \Sigma^*$. We assume familiarity with basic concepts in formal language theory and complexity theory (see e.g. (Kozen, 1997; Sipser, 1997)). In particular, we will deal with finite automata. We will also use the following complexity classes:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}.$$

The complexity classes P and NP correspond to problems solvable in polynomial (resp. nondeterminsitic polynomial) time, and are well-known. The complexity classes EXP and NEXP are similar to P and NP, but we allow the algorithm to use exponential time. The complexity classes PSPACE

and EXPSPACE correspond to problems solvable in polynomial (resp. exponential) space. The above inclusions are well-known (cf. (Sipser, 1997)).

2.1 LINEAR TEMPORAL LOGIC

A formula in Linear Temporal Logic (LTL) over the finite alphabet Σ has the following syntax:

$$\varphi ::= \top \mid \bot \mid Q_a(\text{for all } a \in \Sigma) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \,\mathbf{S}\, \varphi \mid \varphi \,\mathbf{U}\, \varphi$$

We define satisfaction of an LTL formula φ on a word $w = a_1 \dots a_n \in \Sigma^+$ at position $i \in [1, n]$, written $w, i \models \varphi$, inductively (omitting $\top$ (true) and $\bot$ (false)):

$w, i \models Q_a$	iff	$a_i = a$ (for all $a \in \Sigma$)
$w, i \models \varphi_1 \wedge \varphi_2$	iff	$w, i \models \varphi_1$ and $w, i \models \varphi_2$
$w, i \models \varphi_1 \vee \varphi_2$	iff	$w, i \models \varphi_1$ or $w, i \models \varphi_2$
$w, i \models \neg\varphi_1$	iff	$w, i \not\models \varphi_1$
$w, i \models \varphi_1 \,\mathbf{S}\, \varphi_2$	iff	for some j with $1 \le j < i$ we have $w, j \models \varphi_2$ and for all k with $j < k < i$ we have $w, k \models \varphi_1$
$w, i \models \varphi_1 \,\mathbf{U}\, \varphi_2$	iff	for some j with $i < j \le n$ we have $w, j \models \varphi_2$ and for all k with $i < k < j$ we have $w, k \models \varphi_1$

Moreover, we define the shortcuts

$$\mathbf{P}\varphi := \top \,\mathbf{S}\, \varphi \qquad \mathbf{F}\varphi := \top \,\mathbf{U}\, \varphi \qquad \mathbf{X}\varphi := \bot \,\mathbf{U}\, \varphi \qquad \mathbf{G}\varphi := \varphi \wedge \neg\mathbf{F}\neg\varphi.$$

An LTL formula recognizes the language $L(\varphi)$ of all words $w \in \Sigma^+$ such that $w, k \models \varphi$, where k is either 1 or $|w|$ depending on whether the first or last position is regarded the *output position* of φ.

Example 1. *The star-free language $(ab)^+$ can be defined in LTL as*

$$Q_a \wedge \boldsymbol{G}\big(Q_a \to \boldsymbol{X}Q_b\big) \wedge \boldsymbol{G}\big((Q_b \wedge \boldsymbol{X}\top) \to \boldsymbol{X}Q_a\big).$$

In words, the first letter is a and at any a-position, the next letter is b. At any b-position that has a successor, the next letter is a.

2.2 MASKED UNIQUE HARD-ATTENTION TRANSFORMERS

Let Σ be a finite alphabet of tokens. A *token embedding* is a function $emb\colon \Sigma \to \mathbb{Q}^d$ for some $d > 0$. A token embedding naturally extends to a homomorphism $\Sigma^* \to (\mathbb{Q}^d)^*$, where $emb(a_1 \dots a_n) = emb(a_1) \dots emb(a_n)$ for $a_1, \dots, a_n \in \Sigma$.

Remark 2. *In the following we define transformers over arbitrary rational numbers since our upper bounds even hold in this setting. We remark that all of our results also hold for the special case of fixed-precision real numbers, i.e., with a constant number of bits for a fixed transformer regardless of the input length. In fact, the lower bounds already hold for integers of fixed precision.*

Attention layer. A *masked unique hard-attention* (UHA) layer of width $r > 0$ is defined by

- three affine transformations $A, B\colon \mathbb{Q}^r \to \mathbb{Q}^r$ and $C\colon \mathbb{Q}^{2r} \to \mathbb{Q}^s$ with rational valued coefficients (i.e., each is of the form $Q\boldsymbol{x} + \boldsymbol{b}$ where matrix Q and vector $\boldsymbol{b}$ have only rational entries),
- a mask predicate $M\colon \mathbb{N} \times \mathbb{N} \to \{\top, \bot\}$, which is defined by $M(i, j) := \top$ (no masking), $M(i, j) := (j < i)$ (strict future masking), or $M(i, j) := (j > i)$ (strict past masking), and
- a tie-breaking function τ selecting one element of a finite non-empty subset of $\mathbb{N}$, which is either defined as $\min$ (leftmost tie-breaking) or $\max$ (rightmost tie-breaking).

We now show how a UHA layer works on a sequence $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n \in \mathbb{Q}^r$ with $n \ge 1$. The *score function* is defined as the dot product $S(\boldsymbol{v}_i, \boldsymbol{v}_j) := \langle A(\boldsymbol{v}_i), B(\boldsymbol{v}_j)\rangle$ for all $i, j \in [1, n]$. For $i \in [1, n]$ let $U_i := \{j \in [1, n] \mid M(i, j)\}$ be the set of unmasked positions and $B_i := \{j \in U_i \mid \forall j' \in U_i\colon S(\boldsymbol{v}_i, \boldsymbol{v}_j) \ge S(\boldsymbol{v}_i, \boldsymbol{v}_{j'})\}$ be the set of unmasked positions that maximize the score function. We define the *attention vector* at position $i \in [1, n]$ as $\boldsymbol{a}_i := \boldsymbol{v}_{\tau(B_i)}$ if $U_i \neq \emptyset$ and $\boldsymbol{a}_i := \boldsymbol{0}$ otherwise. The layer outputs the sequence $C(\boldsymbol{v}_1, \boldsymbol{a}_1), \dots, C(\boldsymbol{v}_n, \boldsymbol{a}_n)$.

ReLU layer. A ReLU layer of width $r > 0$ on input $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{Q}^r$ applies for some $k \in [1, r]$ the ReLU function to the k-th coordinate of each $\boldsymbol{v}_i$, i.e., it outputs the sequence $\boldsymbol{v}'_1, \ldots, \boldsymbol{v}'_n$ where $\boldsymbol{v}'_i := (\boldsymbol{v}_i[1, k-1], \max\{0, \boldsymbol{v}_i(k)\}, \boldsymbol{v}_i[k+1, n])$. [Equivalently, one could instead allow a feed-forward network at the end of an encoder layer (see (Hao et al., 2022; Barceló et al., 2024)).]

Transformer. A *masked unique hard-attention transformer* (UHAT) is a length-preserving function $\mathcal{T}\colon \Sigma^+ \to (\mathbb{Q}^s)^+$ defined by application of a token embedding followed by application of a fixed sequence of UHA layers and ReLU layers of matching width.

Languages accepted by UHATs. To view a UHAT $\mathcal{T}\colon \Sigma^+ \to (\mathbb{Q}^s)^+$ as a language recognizer, we assume that $\mathcal{T}$ is given together with an *acceptance vector* $\boldsymbol{t} \in \mathbb{Q}^s$. The recognized language $L(\mathcal{T})$ is then the set of words on which $\mathcal{T}$ outputs a sequence $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{Q}^s$ with $\langle \boldsymbol{t}, \boldsymbol{v}_k \rangle > 0$, where $k \in [1, n]$ is either 1 or n depending on whether the first or last position is regarded the *output position* of $\mathcal{T}$.

2.3 Boolean RASP

As an intermediate step to prove EXPSPACE-hardness for UHATs, we use Boolean RASP (B-RASP) as introduced by Yang et al. (2024), who showed that B-RASP is expressively equivalent to UHATs. A B-RASP program is defined as follows. Let $w = a_1 \ldots a_n \in \Sigma^+$ be an input word. For every $a \in \Sigma$ there is an *initial* Boolean vector $Q_a \in \{0,1\}^n$ with $Q_a(i) = 1$ iff $a_i = a$ for all $i \in [1, n]$. We number the initial vectors and call them $P_1, \ldots, P_{|\Sigma|}$. We now describe how vector P_{t+1} can be defined from vectors $P_1, \ldots, P_t$ for $t \geq |\Sigma|$.

Position-wise operation. The vector P_{t+1} can be defined by $P_{t+1}(i) := R(i)$ for some Boolean combination $R(i)$ of $\{P_1(i), \ldots, P_t(i)\}$.

Attention operation. The vector P_{t+1} can be defined by either of

$$P_{t+1}(i) := \blacktriangleleft_j \; [M(i,j), S(i,j)]\, V(i,j) : D(i)$$
$$P_{t+1}(i) := \blacktriangleright_j \; [M(i,j), S(i,j)]\, V(i,j) : D(i)$$

where

- $\blacktriangleleft$ indicates leftmost tie-breaking and $\blacktriangleright$ indicates rightmost tie-breaking,
- $M(i,j)$ is a *mask predicate* as in the definition of a UHAT,
- $S(i,j)$ and $V(i,j)$ are Boolean combinations of $\{P_1(i), \ldots, P_t(i)\} \cup \{P_1(j), \ldots, P_t(j)\}$, called *score predicate* and *value predicate*, respectively,
- $D(i)$ is a Boolean combination of $\{P_1(i), \ldots, P_t(i)\}$, called *default value predicate*.

The semantics of an attention operation is as follows. For every $i \in [1, n]$, let

$$j_i := \begin{cases} \min\{j \in [1,n] \mid M(i,j) \text{ and } S(i,j) = 1\}, & \text{for } \blacktriangleleft \\ \max\{j \in [1,n] \mid M(i,j) \text{ and } S(i,j) = 1\}, & \text{for } \blacktriangleright\,. \end{cases}$$

We then define $P_{t+1}(i) := V(i, j_i)$ if j_i exists and $P_{t+1}(i) := D(i)$ otherwise.

A B-RASP program is given by an alphabet Σ and a sequence of position-wise and attention operations. We can view a B-RASP program as a language recognizer by designating one Boolean vector Y as the *output vector* and either the first or last position as the *output position*. Then an input word $w = a_1 \ldots a_n$ is accepted if and only if $Y(k) = 1$, where k is the output position, i.e., $k = 1$ or $k = n$.

2.4 Recurrent Neural Networks (RNN)

We use RNN as language acceptors, as in the work of Merrill et al. (2020); Weiss et al. (2024; 2018). In particular, a *Recurrent Neural Network (RNN)* M can be viewed as a function $g : (\mathbb{R}^d \times \Sigma) \to \mathbb{R}^d$. As in the case of transformers, Σ is also mapped into $\mathbb{R}^{d'}$ (for some d') through a token embedding

function emb (e.g. one-hot encoding) and the function g actually has domain $\mathbb{R}^d \times \mathbb{R}^{d'}$. We have an input vector $\bar{x}_0 \in \mathbb{R}^d$ and a final function $f : \mathbb{R}^d \to \{Acc, Rej\}$, to decide whether a vector $\bar{x} \in \mathbb{R}^d$ is accepting. The semantics of acceptance is the same as that of automata, i.e., given $w = a_1 \cdots a_n \in \Sigma^*$, we compute d-vectors $\bar{x}_1, \ldots, \bar{x}_n$ such that $g(\bar{x}_i, a_{i+1}) = \bar{x}_{i+1}$ for each $i = 0, \ldots, n-1$. The string w is *accepted* by M if $f(\bar{x}_n) = Acc$.

As a computational model, it is realistic to assume RNNs with a fixed precision, i.e., computation is always done over real numbers that can be represented with a constant k number of bits. The details of the actual representation are not important for our analysis. Therefore, the state-space Q of the above RNN can be mapped to d-vectors over $\{0,1\}^k$ (instead of $\mathbb{R}$). The following proposition is now immediate.

Proposition 3. *An RNN $g : (\mathbb{R}^d \times \Sigma) \to \mathbb{R}^d$ with fixed precision k can be represented by a finite automaton with 2^{kd} many states.*

2.5 Size measures and succinctness

Let $\mathcal{R}$ be a finite representation of a language, i.e., in our case a UHAT, LTL formula, finite automaton, RNN, or B-RASP program. We define the size of $\mathcal{R}$, denoted by $|\mathcal{R}|$, as the length of its usual binary encoding. In measuring succinctness of RNN, we put the precision k in unary also as part of the size measure; since we do not want to compare a transformer that uses a fixed precision k and allow an RNN that uses a fixed precision 2^k. Let $\mathcal{C}_1, \mathcal{C}_2$ be classes of finite representations of languages. We say that $\mathcal{C}_1$ can be *exponentially more succinct* than $\mathcal{C}_2$ if for every function $f \in 2^{o(n)}$ there is an $\mathcal{R}_1 \in \mathcal{C}_1$ such that any $\mathcal{R}_2 \in \mathcal{C}_2$ representing the same language as $\mathcal{R}_1$ has size $|\mathcal{R}_2| > f(|\mathcal{R}_1|)$. Similarly, we define *doubly exponentially more succinct* using functions $f \in 2^{2^{o(n)}}$ instead. Intuitively, this means that any translation from $\mathcal{C}_1$ to $\mathcal{C}_2$ incurs, in the worst-case, an (doubly) exponential increase in size.

3 Size of smallest witness via non-emptiness problem

In this section we consider the problem of checking whether the language recognized by a UHAT or B-RASP program is non-empty. In particular, the technique is essentially a simulation of a Turing machine with an $2^{O(n)}$-sized tape (for a given n). As we will see later, there are Turing machines such that the smallest (i.e. shortest) accepted string by the constructed UHAT is of length at least $2^{2^{\Omega(n)}}$.

Example 4. *To illustrate the idea, we describe a B-RASP program that accepts strings of the form*

$$0000a_1\#0001a_2\#0010a_3\#\ldots\#1111a_{2^4}\#$$

where $a_i \in \{a, b, c\}$ such that $(a_j, a_{j+1}) \in H$ for all $1 \leq j < 2^4$. Here, $H := \{(a,b), (b,c), (b,a), (c,b)\}$ is a set of constraints specifying which symbols can be next to each other. For simplicity, we concentrate on the two main conditions: (i) checking that the bit counter is incremented and (ii) checking that the successive symbols are in H. To check (i), we use the following attention operation:

$$C_{+1}(i) := \blacktriangleright_j \, [j < i, Q_{\#}(j)] \bigvee_{k=1}^{4} \Big(\bigwedge_{r=1}^{k-1} \neg C_r(i) \wedge C_r(j) \Big) \wedge C_k(i) \wedge \neg C_k(j) \wedge \Big(\bigwedge_{r=k+1}^{4} C_r(i) \leftrightarrow C_r(j) \Big) : 1$$

Assume i is a $\#$-position. Attention selects the rightmost $\#$-position j left of position i. Let $b_1^i \ldots b_4^i$ and $b_1^j \ldots b_4^j$ be the bit strings directly left of position i and j, respectively. We assume that we already defined $C_k(i) = b_k^i$ and $C_k(j) = b_k^j$ for all $k \in [1,4]$. Then the above value predicate checks that the binary number $b_1^i \ldots b_4^i$ is the number $b_1^j \ldots b_4^j$ incremented by 1. To check (ii), we can use the attention operation

$$M_{\leftarrow}(i) := \blacktriangleright_j \, [j < i, Q_a(j) \vee Q_b(j) \vee Q_c(j)] \bigvee_{(h,h') \in H} Q_h(j) \wedge Q_{h'}(i) : 1.$$

If i is a position of a symbol a_i, attention picks the rightmost position j of a symbol a_j to the left of i and checks with the value predicate that $(a_j, a_i) \in H$.

This allows us to succinctly recognize a language whose smallest string has length exponential in the number of bits of the binary counter. In the following we describe how to extend this idea such that we can reduce an EXPSPACE*-complete problem to non-emptiness of certain B-RASP programs. Intuitively, we place multiple such strings as above on top of each other, creating multiple rows and columns (separated by #). Moreover, we introduce vertical constraints, i.e., between rows, in addition to the horizontal constraints H. Using this technique, we will see in Theorem 15 how B-RASP programs can even succinctly recognize languages whose smallest string has doubly exponential length.*

We prove the following precise complexity bounds:

Theorem 5. *The non-emptiness problem for UHATs and B-RASP programs is* EXPSPACE*-complete.*

We start with the lower bound and show it first for B-RASP programs.

Proposition 6. *The non-emptiness problem for B-RASP programs is* EXPSPACE*-hard.*

For the proof we use the techniques illustrated in Example 4 and reduce from a so-called *tiling problem*. A *tile* is a quadruple $t \in \mathbb{N}_0^4$, where we write $t = \langle \mathit{left}(t), \mathit{up}(t), \mathit{right}(t), \mathit{down}(t) \rangle$. The 2^n*-tiling problem* is defined as follows:

Given: An integer $n > 0$ in unary, a finite set T of tiles, and a tile $t_{\mathit{fin}} \in T$

Question: Do there exist $m > 0$ and a function $\tau \colon \{1, \ldots, 2^n\} \times \{1, \ldots, m\} \to T$ such that

1. $\tau(2^n, m) = t_{\mathit{fin}}$,
2. $\mathit{down}(\tau(i, 1)) = \mathit{up}(\tau(i, m)) = 0$ for all $1 \leq i \leq 2^n$,
3. $\mathit{left}(\tau(1, j)) = \mathit{right}(\tau(2^n, j)) = 0$ for all $1 \leq j \leq m$,
4. $\mathit{right}(\tau(i, j)) = \mathit{left}(\tau(i + 1, j))$ for all $1 \leq i < 2^n$ and $1 \leq j \leq m$, and
5. $\mathit{up}(\tau(i, j)) = \mathit{down}(\tau(i, j + 1))$ for all $1 \leq i \leq 2^n$ and $1 \leq j < m$?

Note that an instance of the 2^n-tiling problem consists of the number n in unary, a set of available tiles, and a dedicated final tile. A configuration of tiles, i.e., a candidate for the function τ, places tiles in an arbitrary number of rows and 2^n columns. That is, given a 2^n-tiling problem instance, configurations have a fixed number of columns but an unbounded number of rows.

The following is shown as Theorem 5 in (Schwarzentruber, 2019) (by choosing $k = 1$):

Proposition 7. *The 2^n-tiling problem is* EXPSPACE*-complete.*

To proof Proposition 6, we construct a B-RASP program of size polynomial in n that accepts an encoding of a configuration of tiles as a sequence of strings, which are of a similar form as in Example 4, if and only if the configuration is a solution of the given 2^n-tiling problem instance. The key observation is that strict future masking with rightmost tie-breaking enables us to check conditions between successive tiles in a row (condition 4) but also between the current tile and the tile at the most recent past occurrence of the same counter value, i.e., in the same column of the previous row, (condition 5). The proof of the next lemma can can be found in Appendix A.

Lemma 8. *Given a 2^n-tiling problem instance, one can construct in time polynomial in n a B-RASP program, whose language is non-empty if and only if the 2^n-tiling problem instance has a solution.*

Lemma 8 reduces the 2^n-tiling problem to the non-emptiness problem for B-RASP programs. Thus, together with Proposition 7, it implies Proposition 6.

We observe that the B-RASP program constructed in Lemma 8 is of a special form, which allows for a polynomial-time translation to UHAT.

Lemma 9. *Given a B-RASP program $P_1, \ldots, P_m$ where every attention operation is of the form*

$$P_{t+1}(i) := \blacklozenge_j \left[M(i, j), S(j) \wedge \bigwedge_{k \in K} P_k(i) \leftrightarrow P_k(j) \right] V(i, j) : D(i),$$

where

- $|\Sigma| \leq t < m$,
- $\blacktriangleleft\blacktriangleright \in \{\blacktriangleleft, \blacktriangleright\}$,
- $S(j)$ *is a Boolean combinations of* $\{P_1(j), \ldots, P_t(j)\}$, *and*
- $K \subseteq \{1, \ldots, t\}$,

one can construct in polynomial time a UHAT that recognizes the same language.

Proof sketch. Note that Boolean operations can easily be simulated using affine transformations and ReLU. For the attention operations we use an attention layer. The value predicates $V(i, j)$ can be simulated by copying the required components of the j-th vector, that was selected by attention, to the i-th vector using the affine transformation whose result is forwarded and computing the result of the Boolean combination with additional ReLU layers. The part $S(j)$ of the score predicates that only depends on j can be simulated using an additional preliminary layer that already computes the result of $S(j)$ at every position j. For the part $\bigwedge_{k \in K} P_k(i) \leftrightarrow P_k(j)$ that checks equality of two binary numbers, we provide a score function that maximizes the attention score if the two binary numbers are equal. The full proof can be found in Appendix A. □

Combining Proposition 7 and Lemmas 8 and 9 yields the EXPSPACE lower bound for UHAT.

Proposition 10. *The non-emptiness problem for UHAT is* EXPSPACE*-hard.*

We observe that the B-RASP program constructed in Lemma 8 only uses strict future masking and rightmost tie-breaking (and can be adapted to only use strict past masking and leftmost tie-breaking). Moreover, the UHAT in Lemma 9 preserves the mask predicate and tie-breaking. Thus, the EXPSPACE lower bound already holds for UHATs that only use strict future masking and rightmost tie-breaking (similar for strict past masking and leftmost tie-breaking).

Corollary 11. *The non-emptiness problem for UHATs, where every layer uses strict future masking and rightmost tie-breaking (resp. strict past masking and leftmost tie-breaking), is already* EXPSPACE*-hard.*

We now prove the upper bounds in Theorem 5. To this end, we first note that any B-RASP program can be converted in exponential time into an LTL formula using the construction from (Yang et al., 2024). In Proposition 13 we prove that the same is true for UHATs, which improves the doubly exponential construction in (Yang et al., 2024) that translates UHATs to B-RASP programs first. This suffices to prove the exponential-space upper bounds in Theorem 5 since non-emptiness of languages given by LTL formulas can be checked in polynomial space (Sistla & Clarke, 1985).

For the translation from UHAT to LTL, we first have to make the crucial observation that the values occurring during the computation of a UHAT are not "too large".

Proposition 12. *The values occurring in the computation of a UHAT $\mathcal{T}$ can be represented with only a polynomial number of bits in the size of $\mathcal{T}$. Thus, rational numbers with fixed precision, that is polynomial in the size of $\mathcal{T}$, are sufficient.*

Proof. Clearly, ReLU layers do not increase the amount of bits needed since only the maximum with 0 is taken. Since our UHAT model is defined over rational numbers, we can represent each value with a binary number for the numerator and a binary number for the denominator. By taking the LCM, we can further assume that the denominators of the numbers in $emb(\Sigma)$ are all equal. Note that the LCM of linearly many numbers of linear bit size can be represented with polynomially many bits. Similarly, for each attention layer we can assume that the denominators of all coefficients of the affine transformation, whose result is forwarded, are equal. This ensure that in the input sequence and after each layer the values have the same denominator. Next we argue that the numerators and denominators of all values that occur in the computation can be represented with a polynomial number of bits. The output of an attention layer is an affine transformation involving two input vectors. Here, the input values are only multiplied with constants, i.e., the values in the output only depend linearly on input values. Since the denominators of the values after multiplication with coefficients are all equal, addition does not incur an increase in bit length of the denominators. Therefore, a repeated application of linearly many attention layers can only lead to numerators and

denominators whose values are at most exponential in the number of layers, i.e., can be represented with polynomially many bits. For the score function we observe that the dot product after an affine transformation only involves two input vectors. Thus, the number of bits needed to represent the result of the score function is still polynomial. A crucial observation is that the results after applying the score function are not forwarded to the next layer, which would introduce an exponential increase in size in the number of layers. □

By Proposition 12, we can already compute the results of the affine transformations and score functions during the construction of the LTL formula. This means that the LTL formula only has to simulate the position-wise behavior of attention layers, i.e., masking and selecting the position of the attention vector, but not the actual computation of values. The proof of the following proposition can be found in Appendix A.

Proposition 13. *Given a UHAT that recognizes a language L, one can construct in exponential time an LTL formula recognizing L.*

We remark that if we start with a UHAT, where every attention layer uses strict future masking and leftmost tie-breaking (resp. strict past masking and rightmost tie-breaking), then the LTL formula constructed in the proof of Proposition 13 only uses the **P** (resp. **F**) operator. It was shown in (Sistla & Clarke, 1985) that the non-emptiness problem for the fragments of LTL that only allow **P** or **F** is NP-complete. Thus, we obtain an improved complexity upper bound for such restricted UHATs.

Corollary 14. *The non-emptiness problem for UHATs, where every attention layer uses strict future masking and leftmost tie-breaking (resp. strict past masking and rightmost tie-breaking), is in* NEXP.

Note that it was already shown in (Jerad et al., 2025) that such restricted UHATs are equally expressive as the LTL fragment with only **P** (resp. **F**). However, the construction by Jerad et al. (2025) from UHAT to the LTL fragments incurs a doubly exponential blow-up, as opposed to our singly exponential translation.

4 Succinctness against other representations of languages

We now study how succinctly transformers can represent languages compared to standard models from formal language theory.

We first compare transformers to LTL. One indication that transformers may be more succinct than LTL comes from Theorem 5, which shows that the non-emptiness problem for UHATs is EXPSPACE-complete, whereas for LTL the corresponding problem is known to be PSPACE-complete. The following result shows that this exponential gap also manifests in terms of succinctness.

Theorem 15. *UHATs can be exponentially more succinct than LTL.*

Proof. We give a family $\{L_n\}_{n\geq 1}$ of languages such that L_n is recognized by a UHAT of size polynomial in n but any LTL formula recognizing L_n has size exponential in n. Let $\mathcal{M}_n$ be a (deterministic) Turing machine that implements a binary counter with 2^n bits, i.e., when initialized with 0^{2^n}, it increments the binary number until it has written 1^{2^n} on its tape and accepts. In particular, $\mathcal{M}_n$ first checks that it has 2^n many 0's written on its tape using an additional n-bit counter. To increment the 2^n-bit counter, $\mathcal{M}_n$ traverses the counter from left to right while flipping every 1 to 0 until it encounters the first 0, which is then flipped to 1. For the initial check, $\mathcal{M}_n$ uses a linear number of states in n and incrementing can be done with a constant-sized Turing machine. Moreover, $\mathcal{M}_n$ uses an exponential number of tape cells in n and the unique accepting run has length at least 2^{2^n}. In (van Emde Boas, 1997) a reduction from Turing machines to tiling problem instances is presented that encodes configurations of Turing machines in its rows and a correct tiling corresponds to a valid execution of the Turing machine. We observe that the $2^{p(n)}$-tiling problem instance $\mathcal{I}_n$, for some polynomial p, constructed from $\mathcal{M}_n$ has size polynomial in n and it has the property that the smallest correct tiling has at least 2^{2^n} many rows. We showed in Lemmas 8 and 9 that there is a UHAT $\mathcal{T}_n$ of size polynomial in the size of $\mathcal{I}_n$ that recognizes encodings of correct tilings of $\mathcal{I}_n$. Thus, $\mathcal{T}_n$ is of size polynomial in n and the smallest accepted word has length at least 2^{2^n}. We let L_n be the language recognized by $\mathcal{T}_n$. Let φ_n be an LTL formula that recognizes L_n. Since

the smallest accepted word by any LTL formula has length at most exponential in the formula size (using an exponential conversion from LTL to finite automata similar to (Vardi & Wolper, 1994)), it follows that the size of φ_n is at least exponential in n. □

Conversely, we can show that there is no language than can be represented by LTL significantly more succinct than by UHATs. Thus, we may even say that UHATs *are* exponentially more succinct than LTL.

Proposition 16. *Given an LTL formula φ, one can construct in polynomial time a UHAT that recognizes the same language as φ.*

Proof sketch. From φ we construct a UHAT $\mathcal{T}$ that on input w outputs in a dedicated component at position $1 \leq i \leq |w|$ a 1 if $w, i \models \varphi$ and a 0 otherwise. Then the claim can be proven by induction. If φ is an atomic formula or a Boolean combination, we can easily define $\mathcal{T}$. If $\varphi = \varphi_1 \mathbf{S} \varphi_2$, we can assume by induction hypothesis that we already computed the truth value of φ_1 and φ_2 at every position, which we use to compute the truth value of $\neg\varphi_1 \vee \varphi_2$. We then use an attention layer with strict future masking and rightmost tie-breaking to get for every position i the maximal position $j < i$ where $\neg\varphi_1 \vee \varphi_2$ holds and output at position i the truth value of φ_2 from position j. The case where $\varphi = \varphi_1 \mathbf{U} \varphi_2$ is similar using strict past masking and leftmost tie-breaking. □

We show next that compared to finite automata, UHATs can be even doubly exponentially more succinct. To see this, take the UHAT $\mathcal{T}_n$ from the proof of Theorem 15 that is of size polynomial in n and the smallest accepted word has length at least 2^{2^n}. Since any automaton recognizing a non-empty language accepts a word of length at most linear in the automaton size, the smallest automaton that recognizes the same language as $\mathcal{T}_n$ has size at least doubly exponential in n.

Theorem 17. *UHATs can be doubly exponentially more succinct than finite automata.*

Conversely, the best known construction from counter-free automata (that are equally expressive as LTL) to LTL incurs an exponential blow-up (Maler & Pnueli, 1990). Thus, together with Proposition 16, we obtain an exponential-time translation from counter-free automata to UHATs. Note that also the translation in (Yang et al., 2024) from counter-free automata to UHATs increases the size exponentially, when using the results by Maler & Pnueli (1990).

Finally, we combine Theorem 17 and Proposition 3 to obtain the following succinctness gap between UHATs and RNNs.

Corollary 18. *UHATs can be exponentially more succinct than RNNs.*

Since RNNs recognize all regular languages, whereas UHATs only recognize the strict subclass of star-free languages, RNNs are strictly more expressive than UHATs. Thus, succinctness in the above corollary should be interpreted with respect to star-free languages.

5 Applications

As a consequence of our results, we can show that reasoning about languages of UHATs (e.g. equivalence, emptiness, universality, etc.) is provably intractable. Contrast this to deterministic finite automata, where all these problems can be done in polynomial time (cf. (Kozen, 1997)). In the following, we show the precise complexity for the *equivalence problem*, i.e., the problem of checking whether two given UHATs recognize the same language. That the universality problem is EXPSPACE-complete can be proven similarly.

Theorem 19. *Equivalence of UHATs is* EXPSPACE*-complete.*

Proof. To prove the lower bound, we reduce from the non-emptiness problem for UHATs, which by Theorem 5 is EXPSPACE-complete. To this end, let $\mathcal{T}$ be a given UHAT and fix a UHAT $\mathcal{T}_0$ that recognizes the empty language. Then we have that $\mathcal{T}$ and $\mathcal{T}_0$ are equivalent if and only if $\mathcal{T}$ recognizes the empty language.

For the upper bound let the UHATs $\mathcal{T}_1$ and $\mathcal{T}_2$ be given. We apply Proposition 13 to turn $\mathcal{T}_1$ and $\mathcal{T}_2$ in exponential time into LTL formulas φ_1 and φ_2, respectively. Now, $\mathcal{T}_1$ and $\mathcal{T}_2$ are equivalent if and

only if φ_1 and φ_2 are equivalent. The latter can be decided in polynomial space (Sistla & Clarke, 1985), which results in an exponential-space algorithm in total. □

6 Concluding remarks

Related work. Our work was inspired by the works of Yang et al. (2024); Barceló et al. (2024); Jerad et al. (2025); Li & Cotterell (2025), which exhibit a close connection between unique-hard attention transformers and star-free regular languages. In particular, these works also exploited the connection to LTL. However, none of these results investigated neither the issue of succinctness nor computational complexity of verification, which we establish in this paper.

Sälzer et al. (2025) investigated the issue of verifying transformers of various precisions. In particular, it was shown that transformers of fixed precision are at least NEXP-hard (i.e. hard for the class of problems solvable by nondeterministic algorithms that run in exponential time). The technique there implies that transformers can be (singly) exponentially more succinct than finite automata. However, no conclusion can be derived as to their succinctness relative to representations like LTL or RNN. Our result substantially improve this by showing that transformers can be doubly exponentially more succinct than automata, and exponentially more succinct than LTL and RNN. In addition, our work assumes a much simpler model in comparison to the results in (Sälzer et al., 2025). In particular, we use unique-hard attention, whereas in (Sälzer et al., 2025) a combination of softmax and hardmax is employed. Finally, our results use positional masking (as employed in (Yang et al., 2024; Jerad et al., 2025; Li & Cotterell, 2025)) — as a simple class of Positional Embeddings (PEs) — in contrast to (Sälzer et al., 2025), which admits arbitrary PEs of fixed precision. It was recently shown in Sälzer et al. (2026) that non-fixed finite precision average-hard attention and softmax attention transformers have undecidable verification, even without PEs.

Succinctness has also been studied in the context of linguistics. For example, according to Zipf's law of abbreviation (Zipf, 1935), frequently occuring concepts tend to have a succinct description. In particular, Hindu-Arabic numeral system — which evolves into our modern numeral system — allows an exponentially more succinct description than the Roman numeral system. According to Zipf's law, the former potentially enables mathematics and computer science as we see today.

Future work. We mention the challenge of developing an automatic tool for analyzing, verifying, and explaining transformers. More broadly, this is an important problem for explainable AI, as thoroughly described in the survey (Huang et al., 2020). In particular, lots of practical advances have been made on verifying feed-forward neural networks (but not transformers) and some practical tools have been developed in the last decade (see also the results of the most recent annual VNN competition (Brix et al., 2024)). Despite the rather high complexity (EXPSPACE-complete), we pose as a challenge to exploit techniques from automated verification (Clarke et al., 2018) (e.g. symbolic techniques, simulation, etc.) to verify transformers in practice. Similarly, since our EXPSPACE-hardness proof requires transformers that encode large counters, another research avenue is to study subclasses of transformers, which cannot encode these and potentially allow a lower complexity of verification. Another key open question concerns the learnability of succinct transformers, which is at the moment still unclear in the light of existing empirical results, e.g., see (Garg et al., 2022; Naim et al., 2025; Huang et al., 2025)). Finally, we note that our results constitute a first step toward understanding how succinct transformers can represent languages compared to other language acceptor models, e.g., fixed finite precision softmax transformers Li & Cotterell (2025), which are overapproximated by UHAT. We leave the study of succinctness of fixed finite precision softmax and average-hard attention transformers as future work. See (Yang et al., 2026) for an initial attempt.

Acknowledgments

We thank David Chiang, Marco Sälzer, Andy Yang and anonymous reviewers for their helpful feedback. Pascal Bergsträßer and Anthony W. Lin are supported by Deutsche Forschungsgemeinschaft (grant number 522843867) and European Union[1] (ERC, LASD, 101089343).

[1] Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

Pablo Barceló, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir V. Podolskii. Logical languages accepted by transformer encoders with hard attention. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=gbrHZq07mq.

Pascal Bergsträßer, Chris Köcher, Anthony Widjaja Lin, and Georg Zetzsche. The power of hard attention transformers on data sequences: A formal language theoretic perspective. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. doi: 10.52202/079017-3067. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/af58a33861ac45472ea1cc5860d2b13e-Paper-Conference.pdf.

Christopher Brix, Stanley Bak, Taylor T. Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results, 2024. URL https://arxiv.org/abs/2412.19985.

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 7654–7664. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.527. URL https://doi.org/10.18653/v1/2022.acl-long.527.

E.M. Clarke, O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model Checking, second edition*. Cyber Physical Systems Series. MIT Press, 2018. ISBN 9780262038836. URL https://books.google.de/books?id=ps-MEAAAQBAJ.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? A case study of simple function classes. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/c529dba08a146ea8d6cf715ae8930cbe-Abstract-Conference.html.

Martin Grohe and Nicole Schweikardt. The succinctness of first-order logic on linear orders. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pp. 438–447. IEEE Computer Society, 2004. doi: 10.1109/LICS.2004.1319638. URL https://doi.org/10.1109/LICS.2004.1319638.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *CoRR*, abs/2312.00752, 2023. doi: 10.48550/ARXIV.2312.00752. URL https://doi.org/10.48550/arXiv.2312.00752.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Trans. Assoc. Comput. Linguistics*, 8:156–171, 2020. doi: 10.1162/TACL_A_00306. URL https://doi.org/10.1162/tacl_a_00306.

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. Assoc. Comput. Linguistics*, 10:800–810, 2022. doi: 10.1162/TACL_A_00490. URL https://doi.org/10.1162/tacl_a_00490.

Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020. ISSN 1574-0137. doi: 10.1016/j.cosrev.2020.100270. URL https://www.sciencedirect.com/science/article/pii/S1574013719302527.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Raj Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A formal framework for understanding length generalization in transformers. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=U49N5V51rU.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 977–996. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-SHORT.76. URL https://doi.org/10.18653/v1/2025.acl-short.76.

Dexter Kozen. *Automata and computability*. Undergraduate texts in computer science. Springer, 1997. ISBN 978-0-387-94907-9. doi: 10.1007/978-1-4612-1844-9.

Jiaoda Li and Ryan Cotterell. Characterizing the expressivity of transformer language models. *CoRR*, abs/2505.23623, 2025. doi: 10.48550/ARXIV.2505.23623. URL https://doi.org/10.48550/arXiv.2505.23623.

Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004.

Oded Maler and Amir Pnueli. Tight bounds on the complexity of cascaded decomposition of automata. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pp. 672–682. IEEE Computer Society, 1990. doi: 10.1109/FSCS.1990.89589. URL https://doi.org/10.1109/FSCS.1990.89589.

Robert McNaughton and Seymour Papert. *Counter-free Automata*. M.I.T. Press, Cambridge, Mass., 1971. ISBN 0262130769.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. A formal hierarchy of RNN architectures. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 443–459. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.ACL-MAIN.43. URL https://doi.org/10.18653/v1/2020.acl-main.43.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=QZgo9JZpLq.

Omar Naim, Jerome Bolte, and Nicholas Asher. Analyzing limits for in-context learning, 2025. URL https://arxiv.org/abs/2502.03503.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *J. Mach. Learn. Res.*, 22:75:1–75:35, 2021. URL https://jmlr.org/papers/v22/20-302.html.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pp. 46–57, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

Marco Sälzer, Eric Alsmann, and Martin Lange. Transformer encoder satisfiability: Complexity and impact on formal reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=VVO3ApdMUE.

Marco Sälzer, Chris Köcher, Alexander Kozachinskiy, Georg Zetzsche, and Anthony Widjaja Lin. The counting power of transformers. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IAFwK6NyrP.

François Schwarzentruber. The complexity of tiling problems. *CoRR*, abs/1907.00102, 2019. URL http://arxiv.org/abs/1907.00102.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *J. Comput. Syst. Sci.*, 50(1):132–150, 1995. doi: 10.1006/JCSS.1995.1013. URL https://doi.org/10.1006/jcss.1995.1013.

Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997. ISBN 978-0-534-94728-6.

A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. doi: 10.1145/3828.3837. URL https://doi.org/10.1145/3828.3837.

Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Theoretical Computer Science. Birkhäuser Boston, MA, 1 edition, 1994. ISBN 978-0-8176-3719-4. doi: 10.1007/978-1-4612-0289-9. URL https://doi.org/10.1007/978-1-4612-0289-9.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? A survey. *Trans. Assoc. Comput. Linguistics*, 12:543–561, 2024. doi: 10.1162/TACL_A_00663. URL https://doi.org/10.1162/tacl_a_00663.

Peter van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, pp. 331–363. CRC Press, 1997. doi: 10.1201/9780429187490. URL https://doi.org/10.1201/9780429187490.

Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi: 10.1006/INCO.1994.1092. URL https://doi.org/10.1006/inco.1994.1092.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pp. 740–745. Association for Computational Linguistics, 2018. doi: 10.18653/V1/P18-2117. URL https://aclanthology.org/P18-2117/.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples (extended version). *Mach. Learn.*, 113(5):2877–2919, 2024. doi: 10.1007/S10994-022-06163-2. URL https://doi.org/10.1007/s10994-022-06163-2.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/13d7f172259b11b230cc5da8768abc5f-Abstract-Conference.html.

Andy Yang, Pascal Bergsträßer, Georg Zetzsche, David Chiang, and Anthony Lin. Length generalization bounds for transformers, 2026. Under submission (preprint: https://zenodo.org/records/18800700).

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua M. Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? A study in length generalization. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=AssIuHnmHX.

George Kingsley Zipf. *The Psychobiology of Language: An Introduction to Dynamic Philology*. Houghton Mifflin, Boston, MA, 1935.

A Proofs from Section 3

A.1 Proof of Lemma 8

Let a 2^n-tiling problem instance with $n > 0$ in unary, a set of tiles T, and a final tile $t_{fin} \in T$ be given. We encode a configuration of tiles, i.e., a candidate for the function τ in the 2^n-tiling problem definition, as a word over the alphabet $\Sigma := T \cup \{0, 1, \#\}$. We define $enc_\tau \colon \{1, \ldots, 2^n\} \times \{1, \ldots, m\} \to \Sigma^*$ such that

$$enc_\tau(i, j) := \langle i - 1 \rangle \tau(i, j) \#$$

for all $i \in [1, 2^n]$ and $j \in [1, m]$, where $\langle i - 1 \rangle$ denotes the binary encoding of $i - 1$ with n bits and most significant bit first. Then

$$enc(\tau) := enc_\tau(1, 1) \ldots enc_\tau(2^n, 1) enc_\tau(2, 1) \ldots enc_\tau(m, 2^n).$$

We construct a B-RASP program that accepts $enc(\tau)$ if and only if τ satisfies the conditions above.

The B-RASP program first checks whether the input is a word from $(\{0, 1\}^n T \#)^*$ using the following Boolean vectors:

$$\begin{aligned}
A_T(i) &:= \blacktriangleright_j\ [j < i, 1]\ \bigvee_{t \in T} Q_t(j) : 0 \\
A_{C,1}(i) &:= \blacktriangleright_j\ [j < i, 1]\ Q_0(j) \vee Q_1(j) : 0 \\
A_{C,k}(i) &:= \blacktriangleright_j\ [j < i, 1]\ A_{C,k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n \\
A_{\#,1}(i) &:= \blacktriangleright_j\ [j < i, 1]\ Q_\#(j) : 1 \\
A_{\#,k}(i) &:= \blacktriangleright_j\ [j < i, 1]\ A_{\#,k-1}(j) : 1 \qquad \text{for } k = 2, \ldots, n+1 \\
A_{enc}(i) &:= \big(Q_\#(i) \to A_T(i)\big) \wedge \Big(\big(\bigvee_{t \in T} Q_t(i)\big) \to \big(\bigwedge_{k=1}^{n} A_{C,k}(i)\big) \wedge A_{\#,n+1}(i)\Big)
\end{aligned}$$

We use the vector

$$A(i) := \blacktriangleright_j\ [j < i, \neg A_{enc}(j)]\ 0 : A_{enc}(i)$$

to check that $A_{enc}(i) = 1$ at every position i, which is the case if and only if $A(\ell) = 1$ where ℓ is the length of the input. Note that we still have to check that the symbol at position ℓ is $\#$. But before that, we ensure that for every two consecutive binary numbers separated by $\#$ the encoded value increases by 1 or is set to 0 if $2^n - 1$ is reached.

$$\begin{aligned}
C_1(i) &:= \blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]\ Q_1(j) : 0 \\
C_k(i) &:= \blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]\ C_{k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n \\
C_{+1}(i) &:= \blacktriangleright_j\ [j < i, Q_\#(j)]\ \bigvee_{k=1}^{n} \big(\bigwedge_{r=1}^{k-1} \neg C_r(i) \wedge C_r(j)\big) \wedge C_k(i) \wedge \neg C_k(j) \wedge \\
&\qquad \big(\bigwedge_{r=k+1}^{n} C_r(i) \leftrightarrow C_r(j)\big) : 0 \\
C_{1\to 0}(i) &:= \blacktriangleright_j\ [j < i, Q_\#(j)]\ \bigwedge_{k=1}^{n} \neg C_k(i) \wedge C_k(j) : \bigwedge_{k=1}^{n} \neg C_k(i) \\
C(i) &:= \blacktriangleright_j\ [j < i, Q_\#(j) \wedge \neg C_{1\to 0}(j) \wedge \neg C_{+1}(j)]\ 0 : C_{1\to 0}(i) \wedge C_{+1}(i)
\end{aligned}$$

Now, $C(\ell) = 1$ if and only if the binary numbers are as required.

Next, we check that the input ends with $1^n t_{fin} \#$.

$$B_t(i) := \blacktriangleright_j \, [j < i, \bigvee_{t' \in T} Q_{t'}(j)] \, Q_t(j) : 0 \qquad \text{for all } t \in T$$

$$F(i) := Q_\#(i) \wedge B_{t_{fin}}(i) \wedge \bigwedge_{k=1}^{n} C_k(i)$$

Then $F(\ell) = 1$ if and only if the input ends with $1^n t_{fin} \#$.

We continue by verifying conditions 2 and 3 of τ.

$$E_\perp(i) := \blacktriangleright_j \, [j < i, Q_\#(j) \wedge \bigwedge_{k=1}^{n} C_k(i) \leftrightarrow C_k(j)] \, 1 : \bigvee_{t \in T:\ down(t)=0} B_t(i)$$

$$E_\top(i) := \blacktriangleright_j \, [j < i, Q_\#(j) \wedge \big((\bigvee_{t \in T:\ up(t) \neq 0} B_t(j)) \vee \bigwedge_{k=1}^{n} \neg C_k(j) \big)] \, \big(\bigvee_{t \in T:\ up(t)=0} B_t(j) \big) \wedge$$
$$\big(\bigvee_{t \in T:\ up(t)=0} B_t(i) \big) : 0$$

$$E_\vdash(i) := \big(\bigwedge_{k=1}^{n} \neg C_k(i) \big) \rightarrow \big(\bigvee_{t \in T:\ left(t)=0} B_t(i) \big)$$

$$E_\dashv(i) := \big(\bigwedge_{k=1}^{n} C_k(i) \big) \rightarrow \big(\bigvee_{t \in T:\ right(t)=0} B_t(i) \big)$$

$$E(i) := \blacktriangleright_j \, [j < i, Q_\#(j) \wedge \neg(E_\perp(j) \wedge E_\vdash(j) \wedge E_\dashv(j))] \, 0 : E_\perp(i) \wedge E_\top(i) \wedge E_\vdash(i) \wedge E_\dashv(i)$$

Now, conditions 2 and 3 hold if and only if $E(\ell) = 1$.

Finally, we ensure that conditions 4 and 5 are satisfied.

$$M_\downarrow(i) := \blacktriangleright_j \, [j < i, Q_\#(j) \wedge \bigwedge_{k=1}^{n} C_k(i) \leftrightarrow C_k(j)] \bigvee_{t,t' \in T:\ down(t)=up(t')} B_t(i) \wedge B_{t'}(j) : 1$$

$$M_\leftarrow(i) := \blacktriangleright_j \, [j < i, Q_\#(j)] \, \big(\bigvee_{k=1}^{n} C_k(i) \big) \rightarrow \big(\bigvee_{t,t' \in T:\ left(t)=right(t')} B_t(i) \wedge B_{t'}(j) \big) : 1$$

$$M(i) := \blacktriangleright_j \, [j < i, Q_\#(j) \wedge \neg(M_\downarrow(j) \wedge M_\leftarrow(j))] \, 0 : M_\downarrow(i) \wedge M_\leftarrow(i)$$

Then $M(\ell) = 1$ if and only if conditions 4 and 5 hold.

Thus, if we define the output vector to be the conjunction

$$Y(i) := A(i) \wedge C(i) \wedge F(i) \wedge E(i) \wedge M(i)$$

and say that the B-RASP program accepts if and only if $Y(\ell) = 1$, then the B-RASP program recognizes the set of all $enc(\tau)$ where τ satisfies the conditions above. Hence, the language recognized by the B-RASP program is non-empty if and only if the 2^n-tiling problem has a solution.

A.2 Proof of Lemma 9

Let $P_1, \ldots, P_m$ be a B-RASP program over the alphabet $\Sigma = \{a_1, \ldots, a_{|\Sigma|}\}$, where P_t is the initial vector Q_{a_t} for all $1 \leq t \leq |\Sigma|$. We construct a UHAT over Σ that recognizes the same language as the B-RASP program. We use a one-hot token embedding $emb \colon \Sigma \to \{0,1\}^{|\Sigma|}$, i.e., $emb(a_t) := \boldsymbol{e}_t$ for all $1 \leq t \leq |\Sigma|$, where $\boldsymbol{e}_t$ denotes the t-th unit vector. Then $P_t(i)$ coincides with the t-th component of the i-th input vector of the UHAT after the token embedding is applied. The UHAT will preserve these components in each layer and will gradually add new components to store the value of $P_t(i)$ for all $|\Sigma| < t \leq m$. So assume we already defined the layers of the UHAT that

compute the vector $(P_1(i), \ldots, P_t(i))$ at position i for $|\Sigma| \leq t < m$. We now define additional layers whose output will be $(P_1(i), \ldots, P_{t+1}(i))$.

We first consider the case where P_{t+1} is a position-wise operation, i.e., $P_{t+1}(i)$ is defined by a Boolean combination of $\{P_1(i), \ldots, P_t(i)\}$. We define UHAT layers to compute the result of that Boolean combination bottom-up. Assume we already defined layers that output $(R_1(i), \ldots, R_s(i))$, where $s \geq t$ and $R_1(i), \ldots, R_s(i)$ contain $P_1(i), \ldots, P_t(i)$ and the results of previously computed subformulas. To compute the result of $\neg R_k(i)$ for some $k \in [1, s]$, we add an attention layer that just forwards $1 - R_k(i)$ at position i in an additional component while leaving the first s components unchanged. To compute $R_k(i) \wedge R_\ell(i)$ for some $k, \ell \in [1, s]$, we first use an attention layer to forward $R_k(i) + R_\ell(i) - 1$ in an additional component followed by a ReLU layer that forwards the result of $\max\{0, R_k(i) + R_\ell(i) - 1\}$ in this additional component, again leaving the first s components unchanged. We do not have to deal with $R_k(i) \vee R_\ell(i)$, since it can be rewritten as $\neg(\neg R_k(i) \wedge \neg(R_\ell(i))$. After computing the results of all subformulas, we add an additional attention layer to only forward $(P_1(i), \ldots, P_{t+1}(i))$, i.e., removing the intermediate results. Observe that a Boolean combination has only linearly many subformulas.

Let us now consider the case where P_{t+1} is an attention operation of the form

$$P_{t+1}(i) := \blacklozenge_j \left[M(i,j), S(j) \wedge \bigwedge_{k \in K} P_k(i) \leftrightarrow P_k(j)\right] V(i,j) : D(i)$$

as in the statement of Lemma 9. We first use additional layers as in the case of position-wise operations to compute the result of $\neg S(i)$ at every position i in an additional component to output $(P_1(i), \ldots, P_t(i), 1 - S(i))$. Next we use an attention layer to add the result of $1 - P_k(i)$ for all $k \in K$ at every position i in additional components. We then add an attention layer that uses mask predicate M, tie-breaking according to $\blacklozenge$, and attention score

$$\Big(\sum_{k \in K} \big(P_k(i)P_k(j) + (1 - P_k(i))(1 - P_k(j))\big)\Big) - \big(1 - S(j)\big)$$

which is equal to $|\{k \in K \mid P_k(i) = P_k(j)\}| - (1 - S(j))$ since

$$P_k(i)P_k(j) + (1 - P_k(i))(1 - P_k(j)) = \begin{cases} 1, & \text{if } P_k(i) = P_k(j) \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the score is maximized (equal to $|K|$) if $P_k(i) = P_k(j)$ for all $k \in K$ and $S(j) = 1$. For every position i let j_i be the position of the vector that maximizes the attention score w.r.t. i. The attention layer forwards the vector $(P_1(i), \ldots, P_t(i), P_1(j_i), \ldots, P_t(j_i), S(j_i))$ at position i. We now compute the result of

$$R(i) := S(j_i) \wedge \bigwedge_{k \in K} P_k(i) \leftrightarrow P_k(j_i)$$

at every position i as in the case of position-wise operations and forward the vector $(P_1(i), \ldots, P_t(i), P_1(j_i), \ldots, P_t(j_i), R(i))$. Finally, we compute

$$\big(R(i) \wedge V(i, j_i)\big) \vee \big(\neg R(i) \wedge D(i)\big)$$

by again using additional layers as in the case of position-wise operations, whose result is exactly $P_{t+1}(i)$. We then forward $(P_1(i), \ldots, P_{t+1}(i))$.

It remains to describe when the UHAT accepts. If P_t is the output vector of the B-RASP program, then we stop the construction of the UHAT after the layers to compute P_t are constructed. The acceptance vector of the UHAT is then defined as $\boldsymbol{e}_t$, i.e., the t-th unit vector, and the output position k is the same as the output position of the B-RASP program (i.e., the first or last position of the sequence). This means that the UHAT accepts if and only if $\langle \boldsymbol{e}_t, (P_1(k), \ldots, P_t(k))\rangle > 0$, which holds if and only if $P_t(k) = 1$.

We observe that the resulting UHAT only has polynomially many layers since the result of each operation P_t can be computed using an additional number of layers that is linear in the description size of P_t.

A.3 Proof of Proposition 13

Let $\mathcal{T}$ be a UHAT that recognizes a language $L \subseteq \Sigma^+$ and F be a set of binary representations of rational numbers that may occur during the computation of $\mathcal{T}$ from Proposition 12. Our goal is to define for the ℓ-th layer of $\mathcal{T}$ and every vector $\boldsymbol{v} \in F^s$, where s is the output dimension of layer ℓ, an LTL formula $\varphi^\ell_{\boldsymbol{v}}$ such that if $\mathcal{T}$ is applied on input $w \in \Sigma$, then the ℓ-th layer outputs at position $i \in [1, |w|]$ the vector $\boldsymbol{v}$ if and only if $w, i \models \varphi^\ell_{\boldsymbol{v}}$. We define this formula inductively on the layer number ℓ. Let $emb\colon \Sigma \to (\mathbb{Q}^d)^*$ be the token embedding of $\mathcal{T}$. For all $\boldsymbol{v} \in F^d$ let

$$\varphi^0_{\boldsymbol{v}} := \begin{cases} \bigvee_{a \in emb^{-1}(\boldsymbol{v})} Q_a, & \text{if } emb^{-1}(\boldsymbol{v}) \neq \emptyset \\ \bot, & \text{otherwise.} \end{cases}$$

We now define the formula for layer $\ell + 1$. In case of a ReLU layer of width r, that applies ReLU to the k-th coordinate, we can simply define

$$\varphi^{\ell+1}_{\boldsymbol{v}} := \bigvee_{u \in F\colon\ \max\{0,u\} = \boldsymbol{v}[k]} \varphi^\ell_{(\boldsymbol{v}[1,k-1], u, \boldsymbol{v}[k+1,r])}$$

for all $\boldsymbol{v} \in F^r$. If layer $\ell + 1$ is an attention layer with strict future masking and rightmost tie-breaking defined by the affine transformation $C\colon \mathbb{Q}^{2r} \to \mathbb{Q}^s$ and score function $S\colon \mathbb{Q}^{2r} \to \mathbb{Q}^r$, we let

$$\varphi^{\ell+1}_{\boldsymbol{v}} := \bigvee_{\substack{\boldsymbol{u},\boldsymbol{a} \in F^r\colon \\ C(\boldsymbol{u},\boldsymbol{a}) = \boldsymbol{v}}} \varphi^\ell_{\boldsymbol{u}} \wedge \Big(\Big(\bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) < S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \mathbf{S} \Big(\varphi^\ell_{\boldsymbol{a}} \wedge \neg \mathbf{P} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) > S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \Big)$$

for all $\boldsymbol{v} \in F^s$. To account for the special case, where the set of unmasked positions is empty, we take the disjunction of the previous formula and $(\neg \mathbf{P} \top) \wedge \bigvee_{\boldsymbol{u} \in F^r\colon C(\boldsymbol{u},\boldsymbol{0}) = \boldsymbol{v}} \varphi^\ell_{\boldsymbol{u}}$. We omit this special case in the following. If the layer uses leftmost tie-breaking, we adapt the formula as follows:

$$\varphi^{\ell+1}_{\boldsymbol{v}} := \bigvee_{\substack{\boldsymbol{u},\boldsymbol{a} \in F^r\colon \\ C(\boldsymbol{u},\boldsymbol{a}) = \boldsymbol{v}}} \varphi^\ell_{\boldsymbol{u}} \wedge \Big(\mathbf{P}\Big(\varphi^\ell_{\boldsymbol{a}} \wedge \neg \mathbf{P} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) \geq S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \wedge \Big(\neg \mathbf{P} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) > S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \Big)$$

The case of strict past masking is similar, where we use $\mathbf{U}$ instead of $\mathbf{S}$ and $\mathbf{F}$ instead of $\mathbf{P}$. If the layer uses no masking and rightmost tie-breaking, we distinguish three situations: the attention vector is at the current position, the attention vector is strictly to the left of the current position, or the attention vector is strictly to the right of the current position. For the situation, where the attention vector is at the current position, we use

$$\bigvee_{\substack{\boldsymbol{u} \in F^r\colon \\ C(\boldsymbol{u},\boldsymbol{u}) = \boldsymbol{v}}} \varphi^\ell_{\boldsymbol{u}} \wedge \Big(\neg \mathbf{P} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) > S(\boldsymbol{u},\boldsymbol{u})}} \varphi^\ell_{\boldsymbol{b}} \Big) \wedge \Big(\neg \mathbf{F} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) \geq S(\boldsymbol{u},\boldsymbol{u})}} \varphi^\ell_{\boldsymbol{b}} \Big) \tag{1}$$

For the situation, where the attention vector is strictly to the left of the current position, we use

$$\begin{aligned} \bigvee_{\substack{\boldsymbol{u},\boldsymbol{a} \in F^r\colon \\ C(\boldsymbol{u},\boldsymbol{a}) = \boldsymbol{v} \wedge S(\boldsymbol{u},\boldsymbol{a}) > S(\boldsymbol{u},\boldsymbol{u})}} & \varphi^\ell_{\boldsymbol{u}} \wedge \Big(\neg \mathbf{F} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) \geq S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \\ & \wedge \Big(\Big(\bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) < S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \mathbf{S} \Big(\varphi^\ell_{\boldsymbol{a}} \wedge \neg \mathbf{P} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) > S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \Big). \end{aligned} \tag{2}$$

Similarly, for the situation, where the attention vector is strictly to the right of the current position, we use

$$\begin{aligned} \bigvee_{\substack{\boldsymbol{u},\boldsymbol{a} \in F^r\colon \\ C(\boldsymbol{u},\boldsymbol{a}) = \boldsymbol{v} \wedge S(\boldsymbol{u},\boldsymbol{a}) \geq S(\boldsymbol{u},\boldsymbol{u})}} & \varphi^\ell_{\boldsymbol{u}} \wedge \Big(\neg \mathbf{P} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) > S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \\ & \wedge \Big(\mathbf{F}\Big(\varphi^\ell_{\boldsymbol{a}} \wedge \neg \mathbf{F} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) \geq S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \wedge \Big(\neg \mathbf{F} \bigvee_{\substack{\boldsymbol{b} \in F^r\colon \\ S(\boldsymbol{u},\boldsymbol{b}) > S(\boldsymbol{u},\boldsymbol{a})}} \varphi^\ell_{\boldsymbol{b}} \Big) \Big). \end{aligned} \tag{3}$$

Thus, in the case of no masking and rightmost tie-breaking, we define $\varphi^{\ell+1}_{\boldsymbol{v}}$ as the disjunction of Equations (1) to (3). The case where the layer uses no masking and leftmost tie-breaking is analogous.

Finally, if there are m layers, where the last layer outputs vectors of dimension s, and $\boldsymbol{t} \in \mathbb{Q}^s$ is the acceptance vector of $\mathcal{T}$, we define the formula

$$\varphi := \bigvee_{\boldsymbol{v} \in F^s : \langle \boldsymbol{t}, \boldsymbol{v} \rangle > 0} \varphi_{\boldsymbol{v}}^m.$$

Then $w, k \models \varphi$ if and only if $w \in L$, where k is the output position of $\mathcal{T}$.

It remains to argue that φ can be computed in exponential time. By Proposition 12, $|F|$ is exponential in the size of $\mathcal{T}$ and every representation in F is of polynomial size. Moreover, F can be computed in exponential time. The formulas $\varphi_{\boldsymbol{v}}^{\ell+1}$ at every layer $\ell + 1$ of width r depends on $|F|^{O(r)}$ many formulas from layer ℓ. Moreover, $\varphi_{\boldsymbol{v}}^{\ell+1}$ can be computed in time polynomial in $|F|^r \cdot |\mathcal{T}|$, since we only have to compute affine transformations on vectors from F^r, where each component is of size polynomial in $|\mathcal{T}|$. The formulas $\varphi_{\boldsymbol{v}}^m$ at the last layer m depend on $|F|^{O(r'm)}$ many formulas from layer 0, where r' is the maximum width of all layers. Thus, $\varphi_{\boldsymbol{v}}^m$ has size exponential in $|\mathcal{T}|$ and can be computed in exponential time. Therefore, also φ can be computed in exponential time.