Chapter 6

Transformers with unique-hard attention

In this chapter, we consider transformers (encoders) that use *unique-hard attention* (UHATs) instead of the standard *softmax attention*. Intuitively, unique-hard attention always concentrates all attention on a single position. Although this is simplistic compared to the way that transformers actually work, we will see that this assumption makes it possible to exactly characterize what languages unique-hard attention transformers can recognize: depending on the details, they recognize exactly the regular languages in AC^0 or a smaller subset called the *star-free* regular languages. Furthermore, it's possible to prove various facts about them; for example, that it is always possible to increase their expressivity by increasing their depth.

6.1 Background: masked unique-hard attention transformers

Unique-hard attention has been studied in several papers:

- Hahn (2020) introduced UHATs and proved that they cannot recognize PARITY.
- Hao et al. (2022) generalized the above result by showing that UHATs only recognize languages in AC^0.
- Barceló et al. (2024) proved that UHATs can recognize all star-free regular languages.

Here, we look at the results of Angluin et al. (2023), which exactly characterize transformers with rightmost-hard attention and *strict* future masking (which we call MUHATs for short).

6.1.1 Strict masking

Strict future masking means that each position i attends to positions $j < i$. If i is the leftmost position, all positions are masked out, so (following Merrill and Sabharwal (2024)) the attention output is just the zero vector.

6.1.2 Unique-hard attention

For any vector $\mathbf{x} \in \mathbb{R}^d$, define $M(\mathbf{x}) = \{i \in [n] \mid \forall j \in [n], \mathbf{x}[j] \leq \mathbf{x}[i]\}$ to be the set of indices of the maximal elements of $\mathbf{x}$. In *leftmost-hard* attention, the leftmost maximal element is used, replacing Eq. (5.8) with:

$$\alpha[i, j] = \mathbb{I}[j = \min M(\mathbf{s}[i, :])] \tag{6.1}$$

whereas in *rightmost-hard* attention, the rightmost maximal element is used:

$$\alpha[i, j] = \mathbb{I}[j = \max M(\mathbf{s}[i, :])]. \tag{6.2}$$

Leftmost-hard attention was previously called *hard* attention by Hahn (2020) and *unique-hard* attention by Hao et al. (2022). Here, we use the term *unique-hard attention* to refer to either leftmost-hard or rightmost-hard attention.

6.2 Background: star-free languages

Star-free languages are called that because they are described by star-free regular expressions, which we don't need to learn. Instead, we define them in terms of automata and logic.

6.2.1 Counter-free automata

The first definition is counter-free automata. Although we aren't going to use counter-free automata, they are sometimes used in theoretical papers about transformers, and they're probably the best way to intuitively understand what star-free languages are.

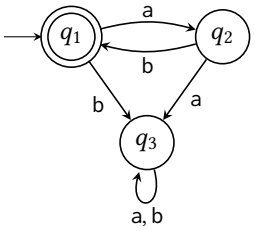
Definition 6.1 (counter-free automaton)**.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Define the relation $q \xrightarrow{\mathbf{w}} r$, where $q, r \in Q$ and $\mathbf{w} \in \Sigma^*$, to mean "If M is in state q and reads string $\mathbf{w}$, then it ends up in state q." That is:

- $q \xrightarrow{\epsilon} q$
- $q \xrightarrow{av} s$ iff, for some r, $\delta(q, a) = r$ and $r \xrightarrow{v} s$.

We say that M is *counter-free* iff there is an N such that for all $\mathbf{w} \in \Sigma^*$ and $n \geq N$, the relations $\xrightarrow{\mathbf{w}^n}$ and $\xrightarrow{\mathbf{w}^{n+1}}$ are the same.

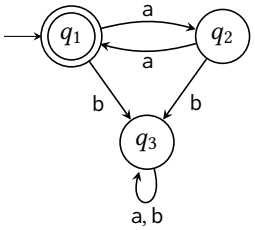
Example 6.2. Intuitively, a counter-free DFA is one that can test whether something happens, but not how many times it happens.

(a) The following DFA, which recognizes $(\mathsf{ab})^*$, is counter-free:



This is counter-free. For any string $\mathbf{w} = (\mathsf{ab})^k$ where $k > 0$, the relation $\xrightarrow{\mathbf{w}}$ sends q_1 to itself and everything else to q_3, and so does $\xrightarrow{\mathbf{ww}}$. Similarly for any string $\mathbf{w} = (\mathsf{ab})^k$. But for any other nonempty string $\mathbf{w}$, $\mathbf{ww}$ must contain a repeated letter, so $\xrightarrow{\mathbf{ww}}$ sends everything to q_3, and so does $\xrightarrow{\mathbf{www}}$.

(b) The following DFA, which recognizes $(\mathsf{aa})^*$, is not counter-free:



This is not counter-free. Let $\mathbf{w} = \mathsf{a}$ and consider the relation $\xrightarrow{\mathbf{w}^n}$. If n is even, it sends q_1 and q_2 to themselves, but if n is odd, it sends q_1 and q_2 to each other.

Theorem 6.3 (Schützenberger, 1965; McNaughton and Papert, 1971)**.** *A language is star-free regular if and only if its minimal DFA is counter-free.*

So in Example 6.2(b), the fact that the DFA shown is minimal and not counter-free shows that $(\mathsf{aa})*$ is not star-free.

6.2.2 First-order logic

A formal language can also be defined as a set of finite strings that satisfy a closed formula of a logic. This section is a very expanded version of Section 5.3 of the survey by Strobl et al. (2024); for more information, see the handbook chapter by Thomas (1997).

Example 6.4. Let $\Sigma = \{\mathsf{a}, \mathsf{b}\}$. The formula

$$\phi = \forall x.\forall y.Q_{\mathsf{a}}(x) \wedge Q_{\mathsf{b}}(y) \rightarrow x < y \tag{6.3}$$

defines the regular language $\mathsf{a}^*\mathsf{b}^*$. The variables (x, y) are interpreted as positions of a string $\mathbf{w}$, and $Q_{\mathsf{a}}(i)$ is true iff $w_i = \mathsf{a}$ and $Q_{\mathsf{b}}(i)$ is true iff $w_i = \mathsf{b}$. So the formula says that every a must precede every b, which is true iff the string matches $\mathsf{a}^*\mathsf{b}^*$.

We first define the syntax of first-order logic with order (FO).

Definition 6.5 (first-order logic with order)**.** The *formulas* of FO are given by the BNF grammar:

$$\begin{aligned} \phi ::= & \; Q_\sigma(x) && \sigma \in \Sigma \\ & \mid x = y \mid x < y \\ & \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \\ & \mid \forall x.\phi_1 \mid \exists x.\phi_1 \end{aligned} \tag{6.4}$$

where $x, y, \ldots$ are variables. The *free variables* of a formula are defined as follows:

$$\begin{aligned} \mathrm{FV}(Q_\sigma(x)) &= \{x\} && \sigma \in \Sigma \\ \mathrm{FV}(x = y) &= \{x, y\} \\ \mathrm{FV}(x < y) &= \{x, y\} \\ \mathrm{FV}(\phi_1 \wedge \phi_2) &= \mathrm{FV}(\phi_1) \cup \mathrm{FV}(\phi_2) \\ \mathrm{FV}(\phi_1 \vee \phi_2) &= \mathrm{FV}(\phi_1) \cup \mathrm{FV}(\phi_2) \\ \mathrm{FV}(\neg\phi_1) &= \mathrm{FV}(\phi_1) \\ \mathrm{FV}(\forall x.\phi_1) &= \mathrm{FV}(\phi_1) \setminus \{x\} \\ \mathrm{FV}(\exists x.\phi_1) &= \mathrm{FV}(\phi_1) \setminus \{x\}. \end{aligned} \tag{6.5}$$

A formula is *closed* if it has no free variables. For example, $\mathrm{FV}(\exists y.x < y) = \{x\}$, while the formula in Eq. (6.3) is closed.

We use a number of shorthand notations:

$$\phi_1 \rightarrow \phi_2 = \neg\phi_1 \vee \phi_2 \qquad \text{implication} \tag{6.6}$$

$$\phi_1 \leftrightarrow \phi_2 = (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1) \qquad \text{if and only if} \tag{6.7}$$

$$\phi_1 \oplus \phi_2 = \neg(\phi_1 \leftrightarrow \phi_2) \qquad \text{exclusive or} \tag{6.8}$$

The semantics of FO defines whether formulas are true in various *logical structures*. Here, we are only interested in logical structures that correspond to strings. So we skip the definition of logical structure and go straight to the definition of whether a formula is true for a string.

Definition 6.6. Let $\mathbf{w} = w_0 \cdots w_{n-1}$ be a string over Σ, and let I be an *interpretation*, or a mapping from variables to $[n]$. We define $\mathbf{w}, I \models \phi$ ("$\mathbf{w}$ and I satisfy ϕ") as follows:

$$\begin{array}{ll}
\mathbf{w}, I \models Q_\sigma(x) & \text{if } w_{I(x)} = \sigma \\
\mathbf{w}, I \models x = y & \text{if } I(x) = I(y) \\
\mathbf{w}, I \models x < y & \text{if } I(x) < I(y) \\
\mathbf{w}, I \models \phi_1 \wedge \phi_2 & \text{if } \mathbf{w}, I \models \phi_1 \text{ and } \mathbf{w}, I \models \phi_2 \\
\mathbf{w}, I \models \phi_1 \vee \phi_2 & \text{if } \mathbf{w}, I \models \phi_1 \text{ or } \mathbf{w}, I \models \phi_2 \\
\mathbf{w}, I \models \neg \phi_1 & \text{if } \mathbf{w}, I \not\models \phi_1 \\
\mathbf{w}, I \models \forall x.\phi_1 & \text{if } \mathbf{w}, I[x \mapsto i] \models \phi_1 \text{ for all } i \in [n] \\
\mathbf{w}, I \models \exists x.\phi_1 & \text{if } \mathbf{w}, I[x \mapsto i] \models \phi_1 \text{ for some } i \in [n]
\end{array} \tag{6.9}$$

Above, the notation $I[x \mapsto i]$ stands for the mapping that sends x to i, and sends any other variable y to $I(y)$.

If $\mathbf{w}, I \models \phi$, and ϕ is a closed formula, we can simply write $\mathbf{w} \models \phi$. The language defined by a closed formula ϕ is $L(\phi) = \{\mathbf{w} \in \Sigma^* \mid \mathbf{w} \models \phi\}$.

Example 6.7. Let ϕ be as in Eq. (6.3):

$$\phi = \forall x.\forall y.Q_{\mathsf{a}}(x) \wedge Q_{\mathsf{b}}(y) \rightarrow x < y.$$

Here are some examples of strings and interpretations that do or don't satisfy ϕ:

$$\begin{aligned}
\mathsf{abb}, \{x \mapsto 0\} &\models Q_{\mathsf{a}}(x) \\
\mathsf{abb}, \{y \mapsto 0\} &\not\models Q_{\mathsf{b}}(y) \\
\mathsf{abb}, \{y \mapsto 1\} &\models Q_{\mathsf{b}}(y) \\
\mathsf{abb}, \{y \mapsto 2\} &\models Q_{\mathsf{b}}(y) \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 0\} &\not\models x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 1\} &\models x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 2\} &\models x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 0\} &\models Q_{\mathsf{a}}(x) \wedge Q_{\mathsf{b}}(y) \rightarrow x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 1\} &\models Q_{\mathsf{a}}(x) \wedge Q_{\mathsf{b}}(y) \rightarrow x < y \\
\mathsf{abb}, \{x \mapsto 0, y \mapsto 2\} &\models Q_{\mathsf{a}}(x) \wedge Q_{\mathsf{b}}(y) \rightarrow x < y \\
\mathsf{abb}, \{x \mapsto 0\} &\models \forall y.Q_{\mathsf{a}}(x) \wedge Q_{\mathsf{b}}(y) \rightarrow x < y
\end{aligned}$$

Example 6.8. As a slightly more complicated example of what can be defined in FO, let

$$\text{FIRST}(x) = \neg\exists y.y < x \tag{6.10}$$
$$\text{SUCC}(x, y) = x < y \wedge \neg\exists z.x < z < y \tag{6.11}$$
$$\text{LAST}(x) = \neg\exists y.y > x \tag{6.12}$$

Then

$$\begin{aligned}\phi = {} & (\forall x.\text{FIRST}(x) \rightarrow Q_{\mathsf{a}}(x)) \\ & \wedge (\forall x.\forall y.\text{SUCC}(x, y) \rightarrow ((Q_{\mathsf{a}}(x) \wedge Q_{\mathsf{b}}(y)) \vee (Q_{\mathsf{b}}(x) \wedge Q_{\mathsf{a}}(y)))) \\ & \wedge (\forall y.\text{LAST}(x) \rightarrow Q_{\mathsf{b}}(x))\end{aligned} \tag{6.13}$$

defines the language $(\mathsf{ab})^*$.

Theorem 6.9 (McNaughton and Papert, 1971)**.** FO *defines exactly the class of star-free regular languages.*

Exercise 6.10. Let $\Sigma = \{\#, 0, 1\}$. Write an FO formula for the language

$$R = \{u\#1v \mid u \in \Sigma^*, v \in (\Sigma \setminus \{\#\})^*\}.$$

Intuitively, a string is in R if the last # is followed by a 1.

6.2.3 Linear temporal logic

In *linear temporal logic* (Kamp, 1968), every formula implicitly depends on a single time (or position). Here, we use a variant that employs only the **since** operator, which is equally as expressive as the complete version (Gabbay et al., 1980).

Example 6.11. Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \#\}$.

- The formula $\phi = Q_{\#}$ defines the language $\Sigma^*\#$, which contains all and only strings with a # in the *last* position.
- The formula $\phi = Q_{\#} \wedge (Q_{\mathsf{b}}$ **since** $Q_{\#})$ defines the language $\Sigma^*\#\mathsf{b}^*\#$. You may read it as, "The last symbol is #, and since the previous #, it's been all b's."
- The formula $\phi = Q_{\#} \wedge (Q_{\mathsf{b}}$ **since** $(Q_{\#} \wedge (Q_{\mathsf{a}}$ **since** $Q_{\#})))$ defines the language $\Sigma^*\#\mathsf{a}^*\#\mathsf{b}^*\#$.
- Finally, the formula

  $$\phi = Q_{\#} \wedge (Q_{\mathsf{b}} \textbf{ since } (Q_{\#} \wedge (Q_{\mathsf{a}} \textbf{ since } (Q_{\#} \wedge \neg(0 \textbf{ since } 1)))))$$

  defines the language $\#\mathsf{a}^*\#\mathsf{b}^*\#$, because $\neg(0$ **since** $1)$ is only true at the leftmost position.

Definition 6.12 (linear temporal logic)**.** The syntax of LTL is defined as follows:

$$\begin{aligned} \phi ::= & \; Q_\sigma && \sigma \in \Sigma \\ & \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg \phi_1 \\ & \mid \phi_1 \textbf{ since } \phi_2 \end{aligned} \tag{6.14}$$

For any input string $\mathbf{w} = w_0 \cdots w_{n-1}$ and position $i \in [n]$, we define $\mathbf{w}, i \models \phi$ as follows:

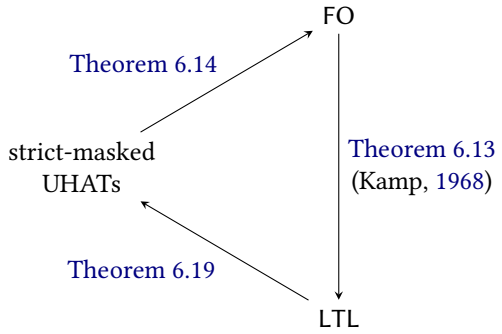
$$\begin{aligned} &\mathbf{w}, i \models Q_\sigma && \text{if } w_i = \sigma \\ &\mathbf{w}, i \models \phi_1 \wedge \phi_2 && \text{if } \mathbf{w}, i \models \phi_1 \text{ and } \mathbf{w}, i \models \phi_2 \\ &\mathbf{w}, i \models \phi_1 \vee \phi_2 && \text{if either } \mathbf{w}, i \models \phi_1 \text{ or } \mathbf{w}, i \models \phi_2 \\ &\mathbf{w}, i \models \neg \phi_1 && \text{if } \mathbf{w}, i \not\models \phi_1 \\ &\mathbf{w}, i \models \phi_1 \textbf{ since } \phi_2 && \text{if there is a } j < i \text{ such that } \mathbf{w}, j \models \phi_2, \text{ and} \\ & && \quad \text{for all } k \text{ such that } j < k < i, \text{ we have } \mathbf{w}, k \models \phi_1 \end{aligned} \tag{6.15}$$

To use a formula ϕ of LTL to define a language over Σ, for a $\mathbf{w} \in \Sigma^*$ of length n we supply $\mathbf{w}$ as input and designate the last position as the output position, so that $\mathbf{w} \in \mathcal{L}(\phi)$ if and only if $\mathbf{w}, n-1 \models \phi$.

Theorem 6.13 (Kamp 1968; Gabbay et al. 1980)**.** LTL *defines the exactly the class of star-free regular languages.*

6.3 Main result

We're going to prove the equivalence differently from Angluin et al. (2023):



Theorem 6.14. *For any transformer encoder T with rightmost hard attention and strict future masking, there is a closed formula of* FO *that defines the same language that T recognizes.*

The proof hinges on the fact that in a unique hard attention transformer, each activation vector depends on at most two vectors from the layer below. Because the network has fixed, finite depth, there is a fixed, finite number of possible activation vectors that it can compute.

Lemma 6.15. *Let T be a unique (leftmost or rightmost) hard attention transformer. There is a finite set $\mathbb{F} \subseteq \mathbb{R}^d$ such that for any input string $\mathbf{w}$, all the activation vectors computed by $T(\mathbf{w})$ belong to $\mathbb{F}$.*

Proof. We prove that the self-attention at layer ℓ has at most $|\Sigma|^{2^\ell}$ different possible output vectors, by induction on ℓ.

Base case ($\ell = 0$): Since there are no position embeddings, the embedding at position i is determined entirely by w_i, so there are at most $|\Sigma| = |\Sigma|^{2^0}$ possible activation vectors.

Inductive step ($\ell > 0$): Assume that the output of the layer $(\ell - 1)$ has at most $|\Sigma|^{2^{\ell-1}}$ possible activation vectors, and consider layer ℓ:

- The attention output at position i depends only on $\mathbf{A}^{(\ell)}[i]$ (because of the residual connection) and $\mathbf{A}^{(\ell)}[j_i]$ (where j_i is the position that i attends to). At position i, there are $|\Sigma|^{2^{\ell-1}}$ possibilities, and at position j_i, there are likewise $|\Sigma|^{2^{\ell-1}}$ possibilities. (If $i = 0$, then there's only $1 \leq |\Sigma|^{2^{\ell-1}}$ possibility.) So the number of possible output activation vectors is at most
$$\left(|\Sigma|^{2^{\ell-1}}\right)^2 = |\Sigma|^{2^\ell}.$$
- Because the FFNN and layernorms operate position-wise, they also have at most $|\Sigma|^{2^\ell}$ possible activation vectors.

Therefore, the number of possible output vectors from a self-attention or FFNN at layer ℓ is at most $|\Sigma|^{2^\ell}$.

Then $\mathbb{F}$ is the union over all layers of the possible activation vectors. □

Although it's not the most efficient way to do it, we can represent each value computed by the network as a set of $|\mathbb{F}|$ many formulas. (Angluin et al. (2023) show how to do this with only $O(\log |\mathbb{F}|)$ many formulas.) Then any positionwise function can be defined by writing a formula that is essentially a lookup table.

Definition 6.16. Let $\mathbb{F}$ be any finite set. A function $\mathbf{X}\colon \Sigma^* \xrightarrow{\text{lp}} \mathbb{F}^*$ is definable by FO formulas $(\phi_{\mathbf{X}=\mathbf{v}}(i))_{\mathbf{v}\in\mathbb{F}}$ if for all $\mathbf{w} \in \Sigma^*$ and $\mathbf{v} \in \mathbb{F}$, we have $\mathbf{X}(\mathbf{w})[i] = \mathbf{v}$ iff $\mathbf{w} \models \phi_{\mathbf{X}=\mathbf{v}}(i)$.

Lemma 6.17. *Let $\mathbb{F}$ and $\mathbb{F}'$ be any finite sets. If $\mathbf{X}\colon \Sigma^* \xrightarrow{\text{lp}} \mathbb{F}^*$ is definable by* FO *formulas $(\phi_{\mathbf{X}=\mathbf{v}}(i))_{\mathbf{v}\in\mathbb{F}}$, and $f\colon \mathbb{F} \to \mathbb{F}'$, then there are* FO *formulas $(\phi_{f(\mathbf{X})=\mathbf{v}}(i))_{\mathbf{v}\in\mathbb{F}}$ that define $\mathbf{w} \mapsto f(\mathbf{X}(\mathbf{w}))$, where f is applied positionwise.*

Proof. For all $\mathbf{v} \in \mathbb{F}'$, define

$$\phi_{f(\mathrm{X})=\mathbf{v}}(i) = \bigvee_{\substack{\mathbf{u}\in\mathbb{F} \\ f(\mathbf{u})=\mathbf{v}}} \phi_{\mathbf{X}=\mathbf{u}}(i). \tag{6.16}$$

□

Proof of Theorem 6.14. For every activation value $\mathbf{A}^{(\ell)}[i]$ (which depends on $\mathbf{w}$ and can be thought of as a function of $\mathbf{w}$), we will construct a formula $\mathrm{act}_{\ell,\mathbf{v}}(i)$ such that $\mathbf{w} \models \mathrm{act}_{\ell,\mathbf{v}}(i)$ iff $\mathbf{A}^{(\ell)}[i] = \mathbf{v}$. We do this by induction on ℓ.

Case $\ell = 0$: At the bottom of the network, we have the embedding layer, $\mathbf{A}^{(0)}[i] = \mathrm{WE}(\mathbf{w}[i])$. We define this in FO as

$$\mathrm{act}_{0,\mathbf{v}}(i) = \bigvee_{\substack{a\in\Sigma \\ \mathrm{WE}(a)=\mathbf{v}}} Q_a(i). \tag{6.17}$$

Case $\ell > 0$: Assume that there are formulas $\mathrm{act}_{\ell-1,\mathbf{v}}(i)$ that define the first ℓ layers, ending in activations $\mathbf{A}^{(\ell-1)}[i]$ (Eq. (5.19) or Eq. (5.20)). Our goal is to write formulas that define $\mathbf{A}^{(\ell)}[i]$. We first need to translate the self-attention into FO, starting with Eq. (5.7) with $\mathbf{X} = \mathbf{A}^{(\ell-1)}$.

By Lemma 6.17, there are formulas $\mathrm{query}_{\ell,\mathbf{v}}(i)$, $\mathrm{key}_{\ell,\mathbf{v}}(i)$, and $\mathrm{value}_{\ell,\mathbf{v}}(i)$ such that

$$\mathbf{w} \models \mathrm{query}_{\ell,\mathbf{v}}(i) \quad \text{iff} \quad \mathbf{W}^{\mathrm{Q}}(\mathbf{A}^{(\ell-1)})[i] = \mathbf{v} \tag{6.18}$$

$$\mathbf{w} \models \mathrm{key}_{\ell,\mathbf{v}}(i) \quad \text{iff} \quad \mathbf{W}^{\mathrm{K}}(\mathbf{A}^{(\ell-1)})[i] = \mathbf{v} \tag{6.19}$$

$$\mathbf{w} \models \mathrm{value}_{\ell,\mathbf{v}}(i) \quad \text{iff} \quad \mathbf{W}^{\mathrm{V}}(\mathbf{A}^{(\ell-1)})[i] = \mathbf{v}. \tag{6.20}$$

By analogy with Lemma 6.17, there are formulas $\mathrm{score}_{\ell,v}(i,j)$ that define $\mathbf{s}^{(\ell)}[i,j]$, the attention score at layer ℓ from position i to position j:

$$\mathrm{score}_{\ell,s}(i,j) = \bigvee_{\substack{\mathbf{u},\mathbf{v}\in\mathbb{F} \\ \mathbf{u}\cdot\mathbf{v}=s}} \Big(\mathrm{query}_{\ell,\mathbf{u}}(i) \wedge \mathrm{key}_{\ell,\mathbf{v}}(j)\Big) \tag{6.21}$$

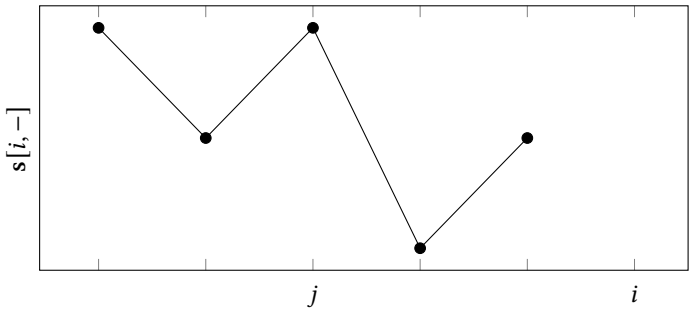
where s ranges over possible scores (a finite set). Furthermore, there is a formula $\mathrm{compare}_\ell(i,j_1,j_2)$ that holds iff $\mathbf{s}^{(\ell)}[i,j_1] \le \mathbf{s}^{(\ell)}[i,j_2]$:

$$\mathrm{compare}_\ell(i,j_1,j_2) = \bigvee_{\substack{s_1,s_2 \\ s_1\le s_2}} (\mathrm{score}_{\ell,s_1}(i,j_1) \wedge \mathrm{score}_{\ell,s_2}(i,j_2)). \tag{6.22}$$

Then we can define a formula that tests whether position i attends to position j (that is, $\mathrm{weight}_\ell(i,j)$ holds iff $\alpha[i,j] = 1$ at layer ℓ).

$$\begin{aligned} \mathrm{weight}_\ell(i,j) = {} & j < i \\ & \wedge \forall k[k < j \rightarrow \mathrm{compare}_\ell(i,k,j)] \\ & \wedge \forall k[j < k \wedge k < i \rightarrow \neg\mathrm{compare}_\ell(i,j,k)]. \end{aligned} \tag{6.23}$$

Because j is the rightmost position with maximal score, the positions left of j must have score less than or equal to $\mathbf{s}[i, j]$, but the positions between j and i (exclusive) must have score strictly less than $\mathbf{s}[i, j]$. For example, if the scores are as shown below, then position j is the one that receives attention:



Next we can write formulas that define the attention output at position i (which is $\mathbf{Y}[i]$ in Eq. (5.10)), taking care to deal with the edge case $i = 0$:

$$\mathrm{att}_{\ell,\mathbf{v}}(i) = \begin{cases} \exists j[\mathrm{weight}_\ell(i, j) \wedge \mathrm{value}_{\ell,\mathbf{v}}(j)] & \mathbf{v} \neq \mathbf{0} \\ \exists j[\mathrm{weight}_\ell(i, j) \wedge \mathrm{value}_{\ell,\mathbf{v}}(j)] \vee \neg\exists j[j < i] & \mathbf{v} = \mathbf{0}. \end{cases} \tag{6.24}$$

By Lemma 6.17 again, there are formulas $\mathrm{act}_{\ell,\mathbf{v}}(i)$ that encode the residual connections and the FFNN (Eq. (5.17) or Eq. (5.18)), defining $\mathbf{A}^{(\ell)}[i]$. This completes the induction.

At the top of the network, we use Lemma 6.17 one last time to write formulas $\mathrm{out}_v(i)$ that define the output layer, and finally we write a formula that tests the output at the last position:

$$\phi = \exists n.(\forall i.i \leq n) \wedge \bigvee_{v \geq 0} \mathrm{out}_v(n). \qquad \square$$

Exercise 6.18. Equation (6.23) is for strictly future-masked, rightmost-hard attention. But the original definition of unique-hard attention (Hahn, 2020) was for unmasked, leftmost-hard attention. Rewrite Eq. (6.23) for unmasked, leftmost-hard attention.

To go in the other direction, we use the fact that LTL is equivalent to FO (Kamp, 1968) and do an easier conversion from LTL.

Theorem 6.19. *For any formula ϕ of* LTL*, there is a transformer encoder with rightmost hard attention and strict future masking that recognizes the same language that ϕ defines.*

The proof will be by induction on the structure of ϕ, so we need the following lemma for combining the translations of sister subformulas.

Lemma 6.20 (Parallel composition)**.** *Given transformer encoders without layer normalization*

$$tfr_1 \colon \Sigma^* \xrightarrow{\text{lp}} (\mathbb{R}^{d_1})^*$$
$$tfr_2 \colon \Sigma^* \xrightarrow{\text{lp}} (\mathbb{R}^{d_2})^*$$

there is a transformer

$$tfr_1 \oplus tfr_2 \colon \Sigma^* \xrightarrow{\text{lp}} (\mathbb{R}^{d_1+d_2})^*$$

such that for all strings $\mathbf{w} = w_0 \cdots w_{n-1} \in \Sigma^*$,

$$(tfr_1 \oplus tfr_2)(\mathbf{w}) = \begin{bmatrix} tfr_1(\mathbf{w})[0] \\ tfr_2(\mathbf{w})[0] \end{bmatrix} \cdots \begin{bmatrix} tfr_1(\mathbf{w})[n-1] \\ tfr_2(\mathbf{w})[n-1] \end{bmatrix}. \tag{6.25}$$

Proof. Let d_1 and d_2 be the width of tfr_1 and tfr_2, and let $d = d_1 + d_2$. If one of tfr_1 and tfr_2 has fewer layers than the other, add trivial layers (layers that compute the identity function) until they have the same number of layers L.

The new transformer has embedding layer

$$(tfr_1 \oplus tfr_2).emb(\mathbf{w}) = \begin{bmatrix} tfr_1.emb(\mathbf{w})[0] \\ tfr_2.emb(\mathbf{w})[0] \end{bmatrix} \cdots \begin{bmatrix} tfr_1.emb(\mathbf{w})[n-1] \\ tfr_2.emb(\mathbf{w})[n-1] \end{bmatrix}.$$

For each layer $\ell \in [L]$, let $f_1 = tfr_1.layer_\ell$ and $f_2 = tfr_2.layer_\ell$. Widen f_1 into a layer f_1' with width d as follows.

$$f_1'.\mathbf{W}^{\mathrm{Q}} = \begin{bmatrix} f_1.\mathbf{W}^{\mathrm{Q}} & \mathbf{0} \end{bmatrix} \qquad f_1'.\mathbf{W}^{\mathrm{K}} = \begin{bmatrix} f_1.\mathbf{W}^{\mathrm{K}} & \mathbf{0} \end{bmatrix}$$
$$f_1'.\mathbf{W}^{\mathrm{V}} = \begin{bmatrix} f_1.\mathbf{W}^{\mathrm{V}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$
$$f_1'.lin_1.\mathbf{W} = \begin{bmatrix} f_1.lin_1.\mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \qquad f_1'.lin_1.\mathbf{b} = \begin{bmatrix} f_1.lin_1.\mathbf{b} \\ \mathbf{0} \end{bmatrix}$$
$$f_1'.lin_2.\mathbf{W} = \begin{bmatrix} f_1.lin_2.\mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \qquad f_1'.lin_2.\mathbf{b} = \begin{bmatrix} f_1.lin_2.\mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

Similarly, widen f_2 into a layer f_2' with width d, but using the bottom half of the

activation vectors:

$$f_2'.\mathbf{W}^{\mathrm{Q}} = \begin{bmatrix} \mathbf{0} & f_2.\mathbf{W}^{\mathrm{Q}} \end{bmatrix} \qquad f_2'.\mathbf{W}^{\mathrm{K}} = \begin{bmatrix} \mathbf{0} & f_2.\mathbf{W}^{\mathrm{K}} \end{bmatrix}$$

$$f_2'.\mathbf{W}^{\mathrm{V}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & f_2.\mathbf{W}^{\mathrm{V}} \end{bmatrix}$$

$$f_2'.lin_1.\mathbf{W} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & f_2.lin_1.\mathbf{W} \end{bmatrix} \qquad f_2'.lin_1.\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ f_2.lin_1.\mathbf{b} \end{bmatrix}$$

$$f_2'.lin_2.\mathbf{W} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & f_2.lin_2.\mathbf{W} \end{bmatrix} \qquad f_2'.lin_2.\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ f_2.lin_2.\mathbf{b} \end{bmatrix}$$

Then stack f_1' on top of f_2', or the other way around. (If we had multi-head attention, we could have combined f_1 and f_2 into a single layer.) □

Proof of Theorem 6.19. For any LTL formula ϕ, there is a transformer tfr_ϕ and an index k such that $tfr_\phi(\mathbf{w})[i,k] = \mathbb{I}[\mathbf{w}, i \models \phi]$. We show this by induction on the structure of ϕ.

Base case $\phi = Q_a$ for some $a \in \Sigma$: Then tfr_ϕ is just an embedding function

$$tfr_\phi(\mathbf{w})[i] = \begin{bmatrix} \mathbb{I}[w_i = a] \end{bmatrix}. \tag{6.26}$$

Case $\phi = \neg\phi_1$: By the induction hypothesis, there is a transformer tfr_1 simulating ϕ_1. For simplicity, we write the output activation vector of tfr_{ϕ_1} as

$$tfr_{\phi_1}(\mathbf{w})[i] = \begin{bmatrix} \mathbb{I}[\mathbf{w}, i \models \phi_1] \\ 0 \end{bmatrix} \tag{6.27}$$

even though $tfr_{\phi_1}(\mathbf{w})[i]$ presumably has other components not shown. We add a trivial self-attention layer and, by Theorem 3.5, a FFNN to simulate $\phi = \neg\phi_1$:

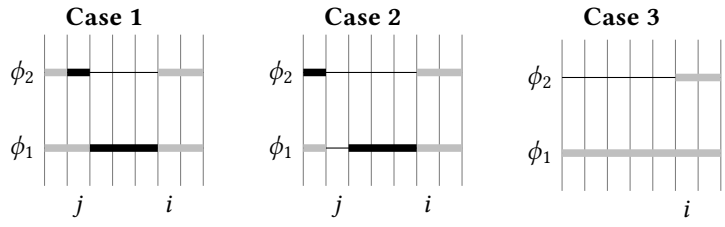
$$ffn_\neg(tfr_{\phi_1}(\mathbf{w}))[i] = \begin{bmatrix} 0 \\ \mathbb{I}[\mathbf{w}, i \models \neg\phi_1] \end{bmatrix}. \tag{6.28}$$

Case $\phi = \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$: By the induction hypothesis, there are transformers tfr_1 and tfr_2 simulating ϕ_1 and ϕ_2, respectively. Combine these using Lemma 6.20, add a trivial self-attention layer and, by Theorem 3.5, a FFNN to simulate ϕ.

Case $\phi = \phi_1$ **since** ϕ_2: By the induction hypothesis, there are transformers tfr_1 and tfr_2 simulating ϕ_1 and ϕ_2, respectively. Combine these using Lemma 6.20. For simplicity, we write the output activation vectors of the composed transformer as:

$$\mathbf{A}[i] = \begin{bmatrix} \mathbb{I}[\mathbf{w}, i \models \phi_1] \\ \mathbb{I}[\mathbf{w}, i \models \phi_2] \\ 0 \\ 0 \end{bmatrix}. \tag{6.29}$$

Imagine, given position i, how you would decide whether ϕ_1 **since** ϕ_2 is true. You could look at positions $i-1, i-2$, and so on. If ϕ_1 is true and ϕ_2 is false, then keep looking to the left. But when you encounter one of these situations, you know whether ϕ_1 **since** ϕ_2 is true (here, black means true, blank means false, and gray means either):



In case 1, the rightmost j satisfying ϕ_2 will occur at or beyond the rightmost j satisfying $\neg\phi_1$, so $w, i \models \phi_1$ **since** ϕ_2. In case 2, the rightmost j satisfying ϕ_2 occurs before the rightmost j satisfying $\neg\phi_1$, so $w, i \not\models \phi_1$ **since** ϕ_2. In case 3, there is no j satisfying ϕ_2, so $w, i \not\models \phi_1$ **since** ϕ_2.

So the idea is to attend to the rightmost position such that ϕ_1 is false or ϕ_2 is true. Then return whether at that position ϕ_2 is true or not. If there is no such position, return false.

Add (a trivial self-attention layer and) a FFNN (using Theorem 3.5) to compute

$$\mathbf{A}'[i] = \begin{bmatrix} \mathbb{I}[\mathbf{w}, i \models \phi_1] \\ \mathbb{I}[\mathbf{w}, i \models \phi_2] \\ \neg\mathbb{I}[\mathbf{w}, i \models \phi_1] \vee \mathbb{I}[\mathbf{w}, i \models \phi_2] \\ 0 \end{bmatrix}. \tag{6.30}$$

Then add a self-attention layer:

$$\mathbf{Q}[i] = \begin{bmatrix} 1 \end{bmatrix} \qquad \mathbf{K}[j] = \begin{bmatrix} \mathbb{I}[\neg(\mathbf{w}, j \models \phi_1) \vee (\mathbf{w}, j \models \phi_2)] \end{bmatrix} \tag{6.31}$$

$$\mathbf{V}[j] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mathbf{w}, j \models \phi_2 \end{bmatrix}. \tag{6.32}$$

For any position i, the position j_i that receives attention is the rightmost one left of i that either satisfies ϕ_2 (in which case ϕ_1 must be satisfied from j_i to i exclusive) or does not satisfy ϕ_1 (in which case ϕ_2 must not be satisfied from j_i to i exclusive).

Then i satisfies (ϕ_1 **since** ϕ_2) if and only if j_i satisfies ϕ_2. So the value tests for whether ϕ_2 is satisfied.

There are two edge cases to consider. First, if there are no positions such that ϕ_1 is false or ϕ_2 is true, then all positions maximize the score. The winner is position $(i-1)$, and at that position, ϕ_2 is false, so the attention outputs false, which is correct. Second, at the very leftmost position ($i = 0$), there are no unmasked positions, so the attention outputs the zero vector (= false), which is correct. □

Since this construction uses only position-independent queries (0 or 1), a perhaps surprising consequence is that every transformer encoder with rightmost hard attention and strict future masking is equivalent to one that uses only position-independent queries.

6.4 Additional results

The close correspondence between MUHATs and LTL not only provides a characterization of the expressive power of (strictly masked unique hard-attention) transformers, but also gives fine-grained insights into what factors contribute to their expressive power. By direct application of known results about LTL, we can better understand the role played by strict masking, positional encodings, and depth.

6.4.1 Stutter-invariance

The choice of strict masking deviates from standard practice, and may appear to be a deficiency, but in the setting of MUHATs, it actually contributes to expressive power. Non-strictness is known to reduce expressivity in LTL (Peled and Wilke, 1997), and we can then show that it reduces expressivity in MUHATs as well. Intuitively, non-strict masked operations are unable to distinguish between consecutive positions that have the same symbol. More formally, a language over Σ is called *stutter-invariant* iff for all $u, v \in \Sigma^*$ and $a \in \Sigma$, $uav \in L$ iff $uaav \in L$. An example of a language that is star-free but not stutter-invariant is $\{a\}$.

Theorem 6.21. *MUHATs with only non-strict masking recognize exactly the stutter-invariant star-free languages.*

Proof. Peled and Wilke (1997) prove that LTL with non-strict operators recognizes exactly the stutter-invariant star-free languages. Non-strict **since** is defined as follows:

$$\mathbf{w}, i \models \phi_1 \textbf{ since } \phi_2 \qquad \text{if there is a } j \leq i \text{ such that } \mathbf{w}, j \models \phi_2, \text{ and}$$
$$\text{for all } k \text{ such that } j < k \leq i, \text{ we have } \mathbf{w}, k \models \phi_1$$

The proofs of Theorems 6.14 and 6.19 may be adapted to use non-strict temporal operators and non-strict masking. Thus, non-strict MUHATs and non-strict LTL are equivalent, and the result follows. □

6.4.2 Regular languages in AC^0

So far we have only considered MUHATs without position embeddings, but the proofs Theorems 6.14 and 6.19 go through for transformers extended with position embeddings with *finite image* (the set $\bigcup_n \{\mathrm{PE}_n(i) \mid i \in [n]\}$ is finite) and FO or LTL extended with additional monadic numerical predicates (predicates with one argument that depend only on n, not $\mathbf{w}$). Here, we consider the case of sinusoidal position embeddings, which are similar to the original position embedding (Vaswani et al., 2017).

For any even d, let us define a *rational sinusoidal positional embedding* with d dimensions to be a position embedding

$$\mathrm{PE}_n(i) = \begin{bmatrix} \sin(2\pi f_0 i) \\ \cos(2\pi f_0 i) \\ \cdots \\ \sin(2\pi f_{d/2-1} i) \\ \cos(2\pi f_{d/2-1} i) \end{bmatrix} \qquad f_0, \ldots, f_{d/2-1} \in \mathbb{Q}.$$

Admittedly, in the original definition, the f_k were not rational.

Corollary 6.22. *MUHATs with rational sinusoidal position embeddings recognize exactly the regular languages in* AC^0 *(that is, regular languages definable by a family of Boolean circuits with polynomial size and constant depth).*

Proof. Let MOD be the collection of predicates $\mathrm{MOD}^r_m(i)$ for all $0 \leq r < m$, which hold just in case $i \equiv r \pmod m$. The regular languages in AC^0 are exactly the languages definable in FO[MOD] or first-order logic with modular predicates (Barrington et al., 1992).

(MUHATs to FO[MOD]) Let PE_n be a sinusoidal positional embedding. Since the f's are rational, PE_n has finite image and is also periodic (that is, there is an integer m such that for all i, i and $i + m$ have the same embedding). So we can adapt the proof Theorem 6.14, which expresses a MUHAT in FO, modifying Eq. (6.17) to

$$\mathrm{act}_{0,\mathbf{v}}(i) = \bigvee_{\substack{a \in \Sigma \\ r \in [m] \\ \mathrm{WE}(a)+\mathrm{PE}(r)=\mathbf{v}}} \left(Q_a(i) \wedge \mathrm{MOD}^r_m(i)\right). \tag{6.33}$$

Thus, transformers with positional embedding PE_n are contained in FO[<, MOD], which are the regular languages in AC^0.

(FO[MOD] to LTL[MOD]) By LTL[MOD], we mean LTL extended with MOD predicates as defined above. The proof of Kamp's theorem (Theorem 6.13) works with these (or any) additional predicates.

(LTL[MOD] to MUHATs) We can use a 2-layer ReLU network to compute MOD^r_m (Chiang et al., 2023, Lemma 20):

$$\begin{aligned} h(i) &= \text{ReLU}\left(\sin 2\pi r/m \sin 2\pi i/m + \cos 2\pi r/m \cos 2\pi i/m - \cos 2\pi/m\right) \\ &= \text{ReLU}(\cos(2\pi(i-r)/m)) \\ \text{MOD}^r_m(i) &= (1 - \cos 2\pi/m)h(i). \end{aligned}$$

Thus LTL[MOD] is contained in the languages recognized by MUHATs with rational sinusoidal position embeddings.

Thus, this class of transformers defines exactly the class of regular languages in AC^0. □

6.4.3 Depth

There is a close relationship between **since** operators and self-attention layers. In fact, the **since**-nesting depth of a formula corresponds to the number of attention layers in the corresponding transformer, assuming we use multi-headed attention layers. We've already defined the *depth* of a MUHAT to be the number of attention layers it has (L).

The *temporal depth* of an LTL formula is defined inductively as follows:

$$\begin{aligned} \text{dp}(Q_a) &= 0 & \text{dp}(\phi \wedge \psi) &= \max(\text{dp}(\phi), \text{dp}(\psi)) \\ \text{dp}(\neg\phi) &= \text{dp}(\phi) & \text{dp}(\phi \ \mathbf{since}\ \psi) &= \max(\text{dp}(\phi), \text{dp}(\psi)) + 1. \end{aligned}$$

Let $\mathsf{MUHAT}(\blacktriangleright F)_k$ be the languages recognizable by multi-head transformers of depth k using only future-masked rightmost-hard attention. Let LTL_k be the languages definable by LTL formulas of depth k.

The proof of Theorem 6.19 can be refined to show that an LTL formula of depth k can be simulated by a transformer of depth k. This requires the use of multi-head attention in order to perform more attention operations in parallel.

Proposition 6.23. *For all $k \geq 0$, $\mathsf{LTL}_k \subseteq \mathsf{MUHAT}(\blacktriangleright F)_k$.*

Proof. We will show that every formula up to depth k can be simulated by a transformer of depth k, by induction on k.

If ϕ is depth 0, it is a Boolean combination of the Q_a predicates. So it can be computed entirely in the word embeddings (cf. Eq. (6.26)):

$$tfr_\phi(\mathbf{w})[i] = \left[\mathbb{I}[\mathbf{w}, i \models \phi]\right]. \tag{6.34}$$

For the inductive step, first note that ϕ is a Boolean combination of formulas $\phi_0, \ldots, \phi_{m-1}$ where each ϕ_i has depth at most $(k+1)$ and is of the form $\phi_i = \phi_{i1}$ **since** ϕ_{i2}, so ϕ_{i1} and ϕ_{i2} have depth at most k. By the induction hypothesis,

there are transformers simulating the ϕ_{i1} and ϕ_{i2}, and we can use the parallel composition lemma (Lemma 6.20) to combine them. We then add a multi-head self-attention layer to simulate each **since** operator in parallel, and then a feed-forward layer to simulate the Boolean combination. Thus, we can simulate ϕ with a transformer of depth k. □

Here, we simulated a MUHAT in LTL indirectly, via simulation in FO and Kamp's theorem. However, we can also perform the simulation directly (Angluin et al., 2023, Theorem 4), making it possible to relate the depth of the MUHAT with the resulting LTL formula:

Proposition 6.24. *For all $k \geq 0$, $\mathsf{MUHAT}(\blacktriangleright F)_k \subseteq \mathsf{LTL}_k$.*

Now that we've related the depth of MUHATs with the depth of LTL formulas, we can apply known results about the LTL depth hierarchy in the setting of transformers. Let $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$; then STAIR_k is the set of strings which, after deleting c's, contain a^k as a substring. For example, abaca is in STAIR_2 but not STAIR_3. It was shown by Etessami and Wilke (2000) that the STAIR language separates the levels of the LTL depth hierarchy:

Theorem 6.25. *For all $k \geq 0$, $\mathit{STAIR}_{k+1} \in \mathsf{LTL}_{k+1} \setminus \mathsf{LTL}_k$. Thus, $\mathsf{LTL}_k \subsetneq \mathsf{LTL}_{k+1}$.*

Using this, we can show the following:

Theorem 6.26. *For all $k \geq 0$, $\mathsf{MUHAT}(\blacktriangleright F)_k \subsetneq \mathsf{MUHAT}(\blacktriangleright F)_{k+1}$. Thus, the depth hierarchy for MUHATs is strict.*

Proof. By composing several results from above:

$$\mathsf{MUHAT}(\blacktriangleright F)_k \subseteq \mathsf{LTL}_k \subsetneq \mathsf{LTL}_{k+1} \subseteq \mathsf{MUHAT}(\blacktriangleright F)_{k+1}.$$ □

So in contrast to feedforward neural networks, where two layers is enough to approximate any function, with MUHATs, it's always possible to increase expressivity by adding more layers.