On The Expressiveness Of Masked Hard-Attention Transformers

Course in Foundations of Neural Networks

Borelli Roberto
borelli.roberto@spes.uniud.it

University of Udine

January 30, 2025

Abstract

This work presents the paper by Yang et al. which characterizes the expressiveness of a particular class of transformers with hard attention, where attention is focused on exactly one position at a time. It is shown how to compile a transformer model into the language B-RASP. Furthermore, it is established that B-RASP is equivalent to star-free languages. The proof proceeds in two directions, employing two distinct characterizations of star-free languages: linear temporal logic over finite traces and cascades of reset automata. This study offers a deeper understanding of the transformer formalism, revealing that (i) both the feed-forward and self-attention sublayers play crucial roles, and (ii) increasing the number of layers in a transformer enhances its expressive power. This latter result contrasts with the universal approximation theorem for standard feedforward neural networks.

Contents

Introduction

Transformers B-RASP

Transformer: assumptions

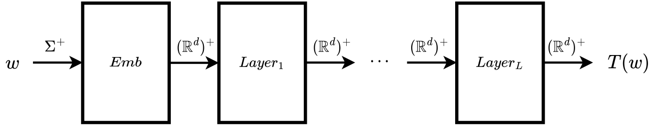
The Transformer studied here:

- is an encoder-only model
- uses **unique hard attention**: all attention is focused on exactly one position

Further simplifications:

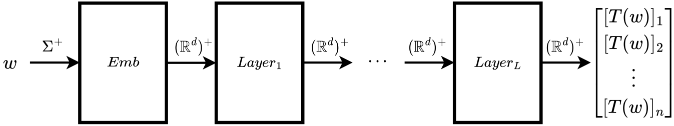
- only single-head attention
- no layer normalization
- no positional embeddings

Transformer: general view



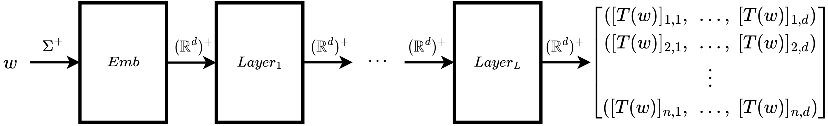
- $T(w)$ is the output on input w
- $[T(w)]_i \in \mathbb{R}^d$ is the $i-$th output vector

Transformer: general view



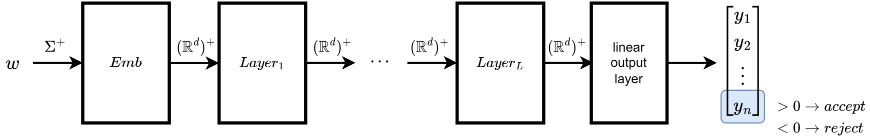
- $T(w)$ is the output on input w
- $[T(w)]_i \in \mathbb{R}^d$ is the $i-$th output vector

Transformer: general view



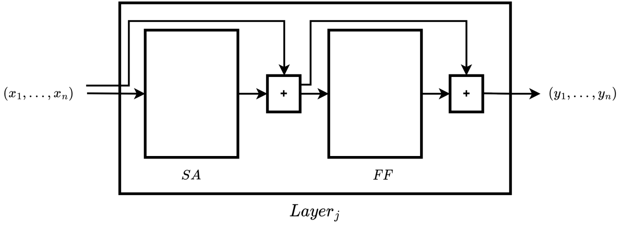
- $T(w)$ is the output on input w
- $[T(w)]_i \in \mathbb{R}^d$ is the $i-$th output vector

Transformer: general view



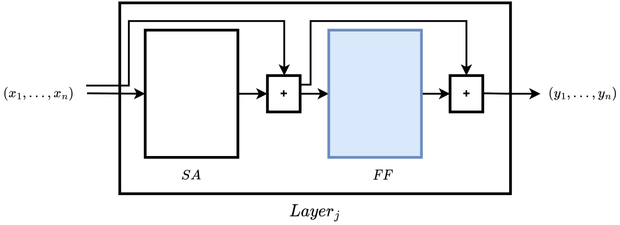
- $T(w)$ is the output on input w
- $[T(w)]_i \in \mathbb{R}^d$ is the $i-$th output vector
- A word w is accepted iff the linear projection of $[T(w)]_n$ is nonnegative

Transformer: layer



$$
\begin{aligned}
Layer_j &: (\mathbb{R}^d)^+ \to (\mathbb{R}^d)^+ \\
(x_1, \ldots, x_n) &\to (y_1, \ldots, y_n)
\end{aligned}
$$

Transformer: layer

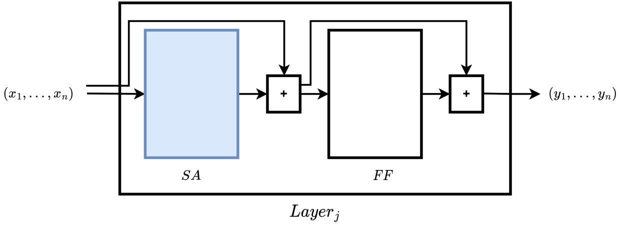


FeedForward SubLayer

It's a feed forward network with two layers and ReLU activation

$$FF(x_i) = (\max\{x_i \cdot W_1 + b_1, 0\})\, W_2 + b_2$$

Transformer: layer



Self Attention SubLayer

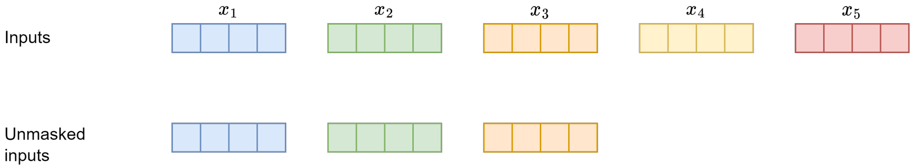
- score function: bilinear function $f_S : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
- value function: linear trasformation $f_V : \mathbb{R}^d \to \mathbb{R}^d$
- mask: one of $M(i,j) = 1$; $M(i,j) = i < j$; $M(i,j) = j < i$
- selector function: either $f_C = \max$ or $f_C = \min$

Example of future-masked leftmost-hard attention

Let $M(i,j) = j < i$ and $f_C = \min$. Let $x_1, \ldots, x_5$ be input vectors. The steps for $SA(x_4)$ are the following.

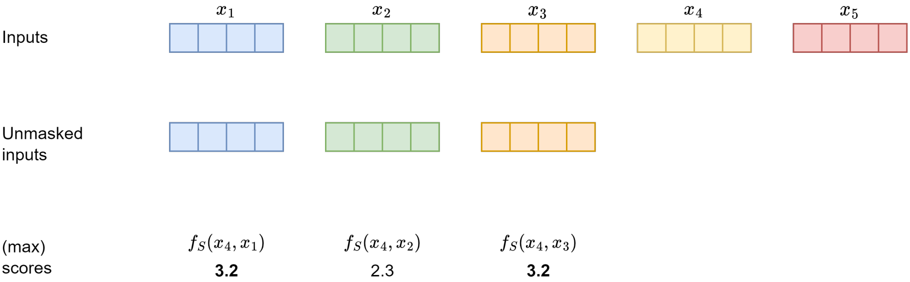
Example of future-masked leftmost-hard attention

Let $M(i,j) = j < i$ and $f_C = \min$. Let $x_1, \ldots, x_5$ be input vectors. The steps for $SA(x_4)$ are the following.



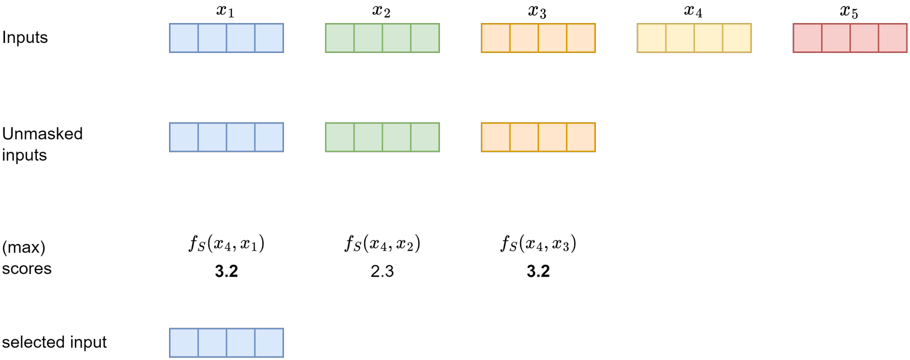
Example of future-masked leftmost-hard attention

Let $M(i,j) = j < i$ and $f_C = \min$. Let $x_1, \ldots, x_5$ be input vectors. The steps for $SA(x_4)$ are the following.



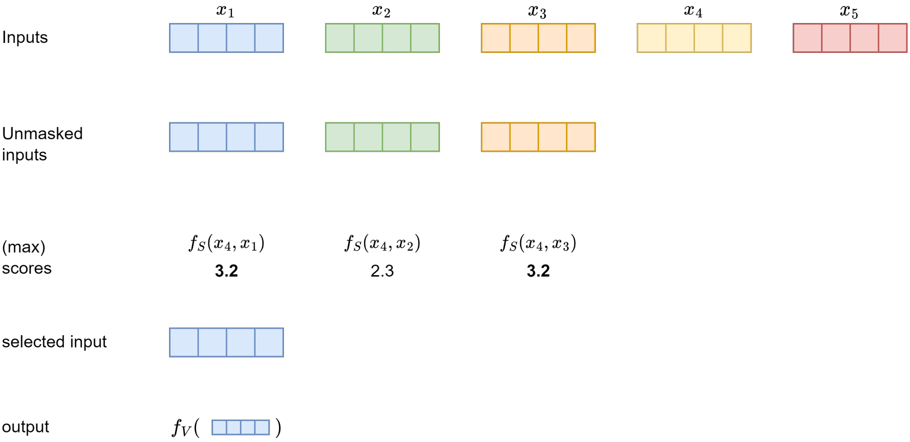
Example of future-masked leftmost-hard attention

Let $M(i,j) = j < i$ and $f_C = \min$. Let $x_1, \ldots, x_5$ be input vectors.
The steps for $SA(x_4)$ are the following.



Example of future-masked leftmost-hard attention

Let $M(i,j) = j < i$ and $f_C = \min$. Let $x_1, \ldots, x_5$ be input vectors. The steps for $SA(x_4)$ are the following.



B-RASP: definition

- It's a programming language intended to help *think like transformers*

B-RASP: definition

- It's a programming language intended to help *think like transformers*

- A B-RASP program p is a sequence of definitions of boolean vectors
- Let $\Sigma = \{\sigma_1, \dots, \sigma_m\}$ be an alphabet and $w = w_1 \dots w_n$ a word in Σ^*

B-RASP: definition

- It's a programming language intended to help *think like transformers*

- A B-RASP program p is a sequence of definitions of boolean vectors
- Let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be an alphabet and $w = w_1 \ldots w_n$ a word in Σ^*
- **Initial vectors:** one-hot encoding of the input

B-RASP: definition

- It's a programming language intended to help *think like transformers*

- A B-RASP program p is a sequence of definitions of boolean vectors
- Let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be an alphabet and $w = w_1 \ldots w_n$ a word in Σ^*
- **Initial vectors:** $P_j(i) = 1 \leftrightarrow w_i = \sigma_j \quad j = 1, \ldots, m$

B-RASP: definition

- It's a programming language intended to help *think like transformers*

- A B-RASP program p is a sequence of definitions of boolean vectors
- Let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be an alphabet and $w = w_1 \ldots w_n$ a word in Σ^*
- **Initial vectors:** $P_j(i) = 1 \leftrightarrow w_i = \sigma_j \quad j = 1, \ldots, m$
- **New vectors:** for $k \geqslant m$ the vector P_{k+1} is defined combining the values of $P_1, \ldots, P_k$ with some operators
- **Output vector:** The last vector is the output vector denoted with Y

B-RASP: definition

- It's a programming language intended to help *think like transformers*
- A B-RASP program p is a sequence of definitions of boolean vectors
- Let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be an alphabet and $w = w_1 \ldots w_n$ a word in Σ^*
- **Initial vectors:** $P_j(i) = 1 \leftrightarrow w_i = \sigma_j \quad j = 1, \ldots, m$
- **New vectors:** for $k \geqslant m$ the vector P_{k+1} is defined combining the values of $P_1, \ldots, P_k$ with some operators
- **Output vector:** The last vector is the output vector denoted with Y
- $w \in \mathcal{L}(p) \leftrightarrow Y(n) = 1$

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \text{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

Attention operations

Syntax $\quad P_{k+1}(i) \coloneqq \text{selector}_j \ [\text{test}] \ V(i,j) \ : \ D(i)$

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

Attention operations

Syntax $P_{k+1}(i) \coloneqq \mathrm{selector}_j \; [\mathrm{test}] \; V(i,j) \; : \; D(i)$

Semantics

```
j' = select j such that test is satisfied
if j' == NULL
    P_{k+1}(i) = D(i)
else
    P_{k+1}(i) = V(i,j')
```

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

Attention operations

Syntax $P_{k+1}(i) \coloneqq \min/\max_j \; [M(i,j) \wedge S(i,j)] \; V(i,j) \; : \; D(i)$

Semantics

```
j' = select min/max j such that M(i,j) ∧ S(i,j) = 1
if j' == NULL
    P_k+1(i) = D(i)
else
    P_k+1(i) = V(i,j')
```

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

Attention operations

$$P_{k+1}(i) \coloneqq \mathrm{min/max}_j \; [M(i,j) \wedge S(i,j)] \; V(i,j) \; : \; D(i)$$

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

Attention operations

$$P_{k+1}(i) \coloneqq \min/\max_j \; [\; \boxed{M(i,j)} \wedge S(i,j)] \; V(i,j) \;\; : \;\; D(i)$$

Mask predicate

$$M(i,j) = \begin{cases} 1 & \text{no masking} \\ j < i & \text{future masking} \\ j > i & \text{past masking} \end{cases}$$

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \dots, P_k(i)\})$$

Attention operations

$$P_{k+1}(i) \coloneqq \mathrm{min/max}_j \ [M(i,j) \wedge \boxed{S(i,j)}\,] \ V(i,j) \ : \ D(i)$$

Score predicate

$$S(i,j) = \mathrm{BoolCombination}(\{P_1(i), \dots, P_k(i)\} \cup \{P_1(j), \dots, P_k(j)\})$$

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

Attention operations

$$P_{k+1}(i) \coloneqq \mathrm{min/max}_j \, [M(i,j) \wedge S(i,j)] \, \boxed{V(i,j)} \;\; : \;\; D(i)$$

Value predicate

$$V(i,j) = \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\} \cup \{P_1(j), \ldots, P_k(j)\})$$

B-RASP: operations

New vectors can be defined in the following way.

Position-wise operations

$$P_{k+1}(i) \coloneqq \mathrm{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$$

Attention operations

$$P_{k+1}(i) \coloneqq \mathrm{min/max}_j \; [M(i,j) \wedge S(i,j)] \; V(i,j) \;\; : \; \boxed{D(i)}$$

Default Value predicate

$$D(i) = \mathrm{BoolCombination}(\{P_1(i), \ldots P_k(i)\})$$

B-RASP: example

Consider $\Sigma = \{a, b\}$, let Q_a and Q_b be the two input vectors. We define the following program p.

$$P_a(i) \coloneqq \max_j [j < i, 1] Q_a(j) : 0$$

$$S_b(i) \coloneqq \min_j [j > i, 1] Q_b(j) : 0$$

$$V \coloneqq (Q_a \wedge S_b) \vee (Q_b \wedge P_a)$$

$$Y \coloneqq \max_j [1, \neg V(j)] 0 : 1$$

It is easy to check that $\mathcal{L}(p) = (ab)^*$

B-RASP: normal form

Lemma (Normalization)

Any B-RASP program is equivalent to one in which all value predicates $V(i,j)$ and all score predicates $S(i,j)$ depend only on j.

Notice that the normalization procedure may require an exponential blowup.

Transformers and B-RASP

Transformers and computed values

Lemma

Let T be a transformer. Let $\mathbb{F}$ be the union over all layers of the possible attention scores and components of the possible activation vectors. It holds that $\mathbb{F}$ is finite.

Transformers and computed values

Lemma

Let T be a transformer. Let $\mathbb{F}$ be the union over all layers of the possible attention scores and components of the possible activation vectors. It holds that $\mathbb{F}$ is finite.

- Any attention score or component of an activation vector, can be represented with $B = \lceil \log_2 \mathbb{F} \rceil$ bits
- For a value v, let $\langle v \rangle_b$ be the bit corresponding to 2^b

Transformers and computed values

Lemma

Let T be a transformer. Let $\mathbb{F}$ be the union over all layers of the possible attention scores and components of the possible activation vectors. It holds that $\mathbb{F}$ is finite.

- Any attention score or component of an activation vector, can be represented with $B = \lceil \log_2 \mathbb{F} \rceil$ bits
- For a value v, let $\langle v \rangle_b$ be the bit corresponding to 2^b

Definition

A B-RASP program p simulates a transformer $T : \Sigma^+ \rightarrow (\mathbb{R}^d)^+$ if

- for each $b \in [B]$ and $k \in [d]$, p contains boolean vectors $Y_{k,b}$
- for every word w of length n, for $i \in [n], b \in [B], k \in [d]$, it holds $Y_{k,b}(i) = 1$ iff $\langle [T(w)]_{i,k} \rangle_b = 1$

Translating transformers to B-RASP

Lemma

For any masked hard-attention transformer T, there is a B-RASP program p_T that simulates T.

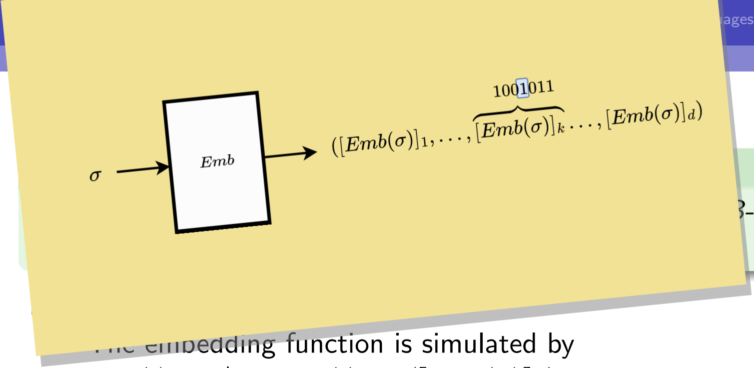
Translating transformers to B-RASP

Lemma

For any masked hard-attention transformer T, there is a B-RASP program p_T that simulates T.

Proof sketch

- The embedding function is simulated by $E_{k,b}(i) \coloneqq \bigwedge_{\sigma \in \Sigma} Q_\sigma(i) \to \langle [Emb(\sigma)]_k \rangle_b$



... -RASP program

The embedding function is simulated by
$E_{k,b}(i) \coloneqq \bigwedge_{\sigma \in \Sigma} Q_\sigma(i) \rightarrow \langle [Emb(\sigma)]_k \rangle_b$

Translating transformers to B-RASP

Lemma

For any masked hard-attention transformer T, there is a B-RASP program p_T that simulates T.

Proof sketch

- The embedding function is simulated by $E_{k,b}(i) \coloneqq \bigwedge_{\sigma \in \Sigma} Q_\sigma(i) \rightarrow \langle [Emb(\sigma)]_k \rangle_b$
- Assume that the first L layers are simulated by p, we extend p to simulate layer $L+1$
- If layer $L+1$ is a FF layer, it's translated into B-RASP using position-wise operations
- If layer $L+1$ is a SA layer, it's translated into B-RASP using attention operations

Translating transformers to B-RASP

Lemma

For any masked hard-attention transformer T, there is a B-RASP program p_T that simulates T.

Proof sketch

- The embedding function is simulated by $E_{k,b}(i) \coloneqq \bigwedge_{\sigma\in\Sigma} Q_\sigma(i) \to \langle [Emb(\sigma)]_k \rangle_b$
- Assume that the first L layers are simulated by p, we extend p to simulate layer $L+1$
- If layer $L+1$ is a FF layer, it's translated into B-RASP using position-wise operations
- If layer $L+1$ is a SA layer, it's translated into B-RASP using attention operations
- Lastly, we add into the program an output vector Y to simulate T's output layer

Translating B-RASP to transformers

Definition

Let p be a B-RASP program with vectors $P_1, \ldots, P_d$. The transformer T (with $d' \geqslant d$ dimensions) simulates p iff for every word w of length n, for all $i \in [n], k \in [d]$ it holds

$$[T(w)]_{i,k} = \begin{cases} 1 & P_k(i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Translating B-RASP to transformers

Definition

Let p be a B-RASP program with vectors $P_1, \ldots, P_d$. The transformer T (with $d' \geqslant d$ dimensions) simulates p iff for every word w of length n, for all $i \in [n], k \in [d]$ it holds

$$[T(w)]_{i,k} = \begin{cases} 1 & P_k(i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Translating B-RASP to transformers

Definition

Let p be a B-RASP program with vectors $P_1, \ldots, P_d$. The transformer T (with $d' \geqslant d$ dimensions) simulates p iff for every word w of length n, for all $i \in [n], k \in [d]$ it holds

$$[T(w)]_{i,k} = \begin{cases} 1 & P_k(i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch

The proof is by induction of the number of vectors of p. Assume that the vectors $P_1, \ldots, P_k$ can be simulated by T_p. We show that P_{k+1} can be simulated as well.

Translating B-RASP to transformers: base cases

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

- Case P_{k+1} is an initial boolean vector for $\sigma \in \Sigma$.
  we set $Emb(\sigma) = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is at position $k+1$

Translating B-RASP to transformers: base cases

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

- Case P_{k+1} is an initial boolean vector for $\sigma \in \Sigma$.
  we set $Emb(\sigma) = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is at position $k + 1$
- Case $P_{k+1}(i)$ is BoolCombination$(\{P_1(i), \ldots, P_k(i)\})$.
  It can be put in DNF and computed by the two layer FFN.

Translating B-RASP to transformers: base cases

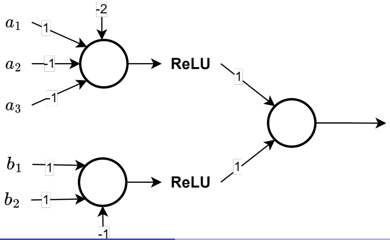
Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

- Case P_{k+1} is an initial boolean vector for $\sigma \in \Sigma$.
  we set $Emb(\sigma) = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is at position $k+1$
- Case $P_{k+1}(i)$ is $\text{BoolCombination}(\{P_1(i), \ldots, P_k(i)\})$.

Example:
$(a_1 \wedge \neg a_2 \wedge \neg a_3) \vee (b_1 \wedge b_2)$



Translating B-RASP to transformers: base cases

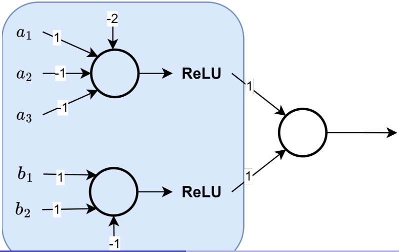
Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

- Case P_{k+1} is an initial boolean vector for $\sigma \in \Sigma$.
  we set $Emb(\sigma) = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is at position $k+1$
- Case $P_{k+1}(i)$ is BoolCombination$(\{P_1(i), \ldots, P_k(i)\})$.

Example:
$(a_1 \wedge \neg a_2 \wedge \neg a_3) \vee (b_1 \wedge b_2)$
The first layer computes conjunctions



Translating B-RASP to transformers: base cases

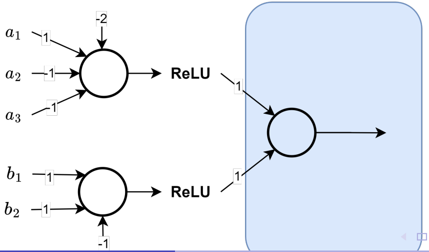
Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

- Case P_{k+1} is an initial boolean vector for $\sigma \in \Sigma$.
  we set $Emb(\sigma) = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is at position $k+1$
- Case $P_{k+1}(i)$ is BoolCombination$(\{P_1(i), \ldots, P_k(i)\})$.

Example:
$(a_1 \wedge \neg a_2 \wedge \neg a_3) \vee (b_1 \wedge b_2)$
The second layer computes disjunctions



Translating B-RASP to transformers: attention operations

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

$$P_{k+1}(i) \coloneqq \min/\max_j \; [M(i,j) \wedge S(i,j)] \; V(i,j) \; : \; D(i).$$

Translating B-RASP to transformers: attention operations

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

$$P_{k+1}(i) \coloneqq \min/\max_j \, [M(i,j) \wedge S(i,j)] \; V(i,j) \;:\; D(i).$$

- Input vector: $(P_1(i), \ldots, P_k(i), 0^{d-k}, 0, 0, \ldots)$

Translating B-RASP to transformers: attention operations

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

$$P_{k+1}(i) \coloneqq \min/\max_j \; [M(i,j) \wedge S(i,j)] \; V(i,j) \; : \; D(i).$$

- Input vector: $(P_1(i), \ldots, P_k(i), 0^{d-k}, 0, 0, \ldots)$
- First layer: The SA sub-layer does nothing (identity function). The FFN adds (at free positions) information in order to compute $S(i,j)$

Translating B-RASP to transformers: attention operations

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

$$P_{k+1}(i) \coloneqq \min/\max_j \, [M(i,j) \wedge S(i,j)] \; V(i,j) \;:\; D(i).$$

- Input vector: $(P_1(i), \ldots, P_k(i), 0^{d-k}, 0, 0, \ldots)$
- First layer: The SA sub-layer does nothing (identity function). The FFN adds (at free positions) information in order to compute $S(i,j)$
- Second Layer: The SA uses mask M and computes either $V(i,j')$ or $D(i)$. The FFN copies the desired result to position $k+1$.

Translating B-RASP to transformers: attention operations

Lemma

For any B-RASP program p, there is a transformer T_p that simulates p.

Proof sketch (cont.)

$$P_{k+1}(i) \coloneqq \min/\max_j \left[M(i,j) \wedge S(i,j)\right] V(i,j) \ : \ D(i).$$

- Input vector: $(P_1(i), \dots, P_k(i), 0^{d-k}, 0, 0, \dots)$
- First layer: The SA sub-layer does nothing (identity function). The FFN adds (at free positions) information in order to compute $S(i,j)$
- Second Layer: The SA uses mask M and computes either $V(i,j')$ or $D(i)$. The FFN copies the desired result to position $k+1$.
- Output vector: either $(P_1(i), \dots, P_k(i), V(i,j'), 0^{d-k-1}, \dots)$ or $(P_1(i), \dots, P_k(i), D(i), 0^{d-k-1}, \dots)$

B-RASP and Star Free Languages

LTL: definition

Syntax

$$
\begin{aligned}
\phi := p \mid \neg\phi \mid \phi \vee \phi & \qquad \text{Boolean operators} \\
\mid \mathsf{X}\phi \mid \phi \,\mathsf{U}\, \phi & \qquad \text{Future operators} \\
\mid \mathsf{Y}\phi \mid \phi \,\mathsf{S}\, \phi & \qquad \text{Past operators}
\end{aligned}
$$

LTL: definition

Syntax

$$\begin{aligned}\phi := p \mid \neg\phi \mid \phi \vee \phi \quad & \text{Boolean operators} \\ \mid \mathsf{X}\phi \mid \phi\,\mathsf{U}\,\phi \quad & \text{Future operators} \\ \mid \mathsf{Y}\phi \mid \phi\,\mathsf{S}\,\phi \quad & \text{Past operators}\end{aligned}$$

Semantics

Let $\mathcal{AP}$ be a set of atomic propositions, and let $\sigma \in (2^{\mathcal{AP}})^{+}$ be a finite trace. Satisfaction of ϕ by σ at time $0 \leqslant i < |\sigma|$ is defined as follows.

LTL: definition

Syntax

$$
\begin{aligned}
\phi := p \mid \neg\phi \mid \phi \vee \phi & \qquad \text{Boolean operators} \\
\mid \mathsf{X}\phi \mid \phi \,\mathsf{U}\, \phi & \qquad \text{Future operators} \\
\mid \mathsf{Y}\phi \mid \phi \,\mathsf{S}\, \phi & \qquad \text{Past operators}
\end{aligned}
$$

Semantics

Let $\mathcal{AP}$ be a set of atomic propositions, and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace. Satisfaction of ϕ by σ at time $0 \leqslant i < |\sigma|$ is defined as follows.

Boolean operators

- $\sigma, i \models p$ iff $p \in \sigma_i$;
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$;
- $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;

LTL: definition

Syntax

$$\begin{aligned} \phi := p \mid \neg\phi \mid \phi \vee \phi & \quad \text{Boolean operators} \\ \mid \mathrm{X}\phi \mid \phi \,\mathrm{U}\, \phi & \quad \text{Future operators} \\ \mid \mathrm{Y}\phi \mid \phi \,\mathrm{S}\, \phi & \quad \text{Past operators} \end{aligned}$$

Semantics

Let $\mathcal{AP}$ be a set of atomic propositions, and let $\sigma \in (2^{\mathcal{AP}})^{+}$ be a finite trace. Satisfaction of ϕ by σ at time $0 \leqslant i < |\sigma|$ is defined as follows.

Future operators

- $\sigma, i \models \mathrm{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models \phi$;
- $\sigma, i \models \phi_1 \,\mathrm{U}\, \phi_2$ iff there exists $i \leqslant j < |\sigma|$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $i \leqslant k < j$;

LTL: definition

Syntax

$$\begin{aligned} \phi := p \mid \neg\phi \mid \phi \vee \phi & \qquad \text{Boolean operators} \\ \mid \mathsf{X}\phi \mid \phi\, \mathsf{U}\, \phi & \qquad \text{Future operators} \\ \mid \mathsf{Y}\phi \mid \phi\, \mathsf{S}\, \phi & \qquad \text{Past operators} \end{aligned}$$

Semantics

Let $\mathcal{AP}$ be a set of atomic propositions, and let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite trace. Satisfaction of ϕ by σ at time $0 \leqslant i < |\sigma|$ is defined as follows.

Past operators

- $\sigma, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\sigma, i-1 \models \phi$;
- $\sigma, i \models \phi_1\, \mathsf{S}\, \phi_2$ iff there exists $j \leqslant i$ such that $\sigma, j \models \phi_2$, and $\sigma, k \models \phi_1$ for all k, with $j < k \leqslant i$;

LTL: shortcuts

We define the following shortcuts.

(True)	$True \coloneqq p \vee \neg p$
(False)	$False \coloneqq \neg True$
(Weak Tomorrow)	$\tilde{X}\phi \coloneqq \neg X \neg \phi$
(Eventually)	$F\phi \coloneqq True \ U \ \phi$
(Globally)	$G\phi \coloneqq \neg F \neg \phi$
(Weak Yesterday)	$\tilde{Y}\phi \coloneqq \neg Y \neg \phi$
(Once)	$O\phi \coloneqq True \ S \ \phi$
(Historically)	$H\phi \coloneqq \neg O \neg \phi$

Translating B-RASP to LTL

Lemma

For any Boolean vector P_k of a B-RASP program in normal form, there is a LTL formula ϕ_k such that for any input w of length n and all $i \in [n]$, it holds $P_k(i) = 1$ iff $w, i \models \phi_k$.

Translating B-RASP to LTL

Lemma

For any Boolean vector P_k of a B-RASP program in normal form, there is a LTL formula ϕ_k such that for any input w of length n and all $i \in [n]$, it holds $P_k(i) = 1$ iff $w, i \models \phi_k$.

Proof

Initial vectors and new vectors defined using point-wise operations are easily translated.

In the following, we show how to perform translation of vectors defined with attention operations starting with *max*. The cases with *min* are analogous.

Translating B-RASP to LTL: future masking

Lemma

For any Boolean vector P_k of a B-RASP program in normal form, there is a LTL formula ϕ_k such that for any input w of length n and all $i \in [n]$, it holds $P_k(i) = 1$ iff $w, i \models \phi_k$.

Proof (cont.)

$$P_k(i) \coloneqq \max_j [j < i, S(j)] V(j) : D(i)$$

is translated to

$$\phi_k \coloneqq Y(\neg\phi_S \; S \; (\phi_S \wedge \phi_V)) \vee ((\tilde{Y} H \; \neg\phi_S) \wedge \phi_D)$$

Formulas ϕ_S, ϕ_V and ϕ_D exist by induction hypothesis.

...re masking

Observation
Future masking is translated using only past operators

...rogram in normal form, there is ...v of length n and all $i \in [n]$, it

$$P_k(i) \coloneqq \max_j [j < i, S(j)] V(j) : D(i)$$

is translated to

$$\phi_k \coloneqq Y(\neg\phi_S \; S \; (\phi_S \wedge \phi_V)) \vee ((\tilde{Y} H \; \neg\phi_S) \wedge \phi_D)$$

Formulas ϕ_S, ϕ_V and ϕ_D exist by induction hypothesis.

Translating B-RASP to LTL: past masking

Lemma

For any Boolean vector P_k of a B-RASP program in normal form, there is a LTL formula ϕ_k such that for any input w of length n and all $i \in [n]$, it holds $P_k(i) = 1$ iff $w, i \models \phi_k$.

Proof (cont.)

$$P_k(i) \coloneqq \max_j [j > i, S(j)] V(j) : D(i)$$

is translated to

$$\phi_k \coloneqq XF(\phi_S \wedge \phi_V \wedge (\tilde{X} G \neg \phi_S)) \vee ((\tilde{X} G \neg \phi_S) \wedge \phi_D)$$

Formulas ϕ_S, ϕ_V and ϕ_D exist by induction hypothesis.

masking

Observation
Past masking is translated using only future operators

rogram in normal form, there is
v of length n and all $i \in [n]$, it

$$P_k(i) \coloneqq \max_j [j > i, S(j)] V(j) : D(i)$$

is translated to

$$\phi_k \coloneqq XF(\phi_S \wedge \phi_V \wedge (\tilde{X} G \neg \phi_S)) \vee ((\tilde{X} G \neg \phi_S) \wedge \phi_D)$$

Formulas ϕ_S, ϕ_V and ϕ_D exist by induction hypothesis.

Translating B-RASP to LTL: no masking

Lemma

For any Boolean vector P_k of a B-RASP program in normal form, there is a LTL formula ϕ_k such that for any input w of length n and all $i \in [n]$, it holds $P_k(i) = 1$ iff $w, i \models \phi_k$.

Proof (cont.)

$$P_k(i) \coloneqq \max_j [1, S(j)] V(j) : D(i)$$

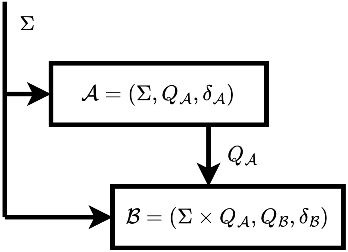
is translated to

$$\begin{aligned} exists(\phi) &\coloneqq (F\phi) \vee (O\phi) \\ rightmost(\phi, \psi) &\coloneqq \phi \wedge \psi \wedge (\tilde{X} G \neg\phi) \\ \phi_k &\coloneqq exists(rightmost(\phi_S, \phi_v)) \vee (\neg exists(\phi_S) \wedge \phi_D) \end{aligned}$$

Formulas ϕ_S, ϕ_V and ϕ_D exist by induction hypothesis. □

Automata Cascades and Krohn-Rhodes theory

Cascade product of automota is the generalization of the direct product

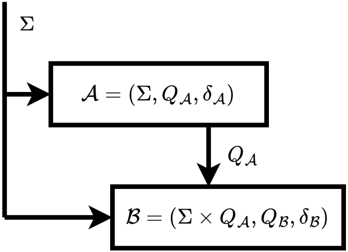


$\mathcal{A} \circ \mathcal{B} = (\Sigma, Q_{\mathcal{A}} \times Q_{\mathcal{B}}, \delta)$ where
$\delta((q_{\mathcal{A}}, q_{\mathcal{B}}), \sigma) = (\delta_{\mathcal{A}}(q_{\mathcal{A}}, \sigma), \delta_{\mathcal{B}}(q_{\mathcal{B}}, (q_{\mathcal{A}}, \sigma)))$

The second automaton reads the current state of the first one

Automata Cascades and Krohn-Rhodes theory

Cascade product of automota is the generalization of the direct product



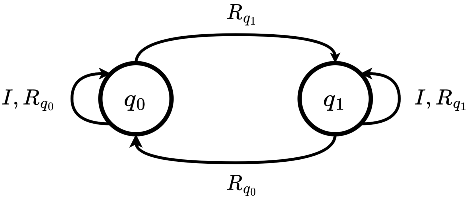
$\mathcal{A} \circ \mathcal{B} = (\Sigma, Q_{\mathcal{A}} \times Q_{\mathcal{B}}, \delta)$ where
$\delta((q_{\mathcal{A}}, q_{\mathcal{B}}), \sigma) = (\delta_{\mathcal{A}}(q_{\mathcal{A}}, \sigma), \delta_{\mathcal{B}}(q_{\mathcal{B}}, (q_{\mathcal{A}}, \sigma)))$

The second automaton reads the current state of the first one

Lemma

Every counter-free automaton $\mathcal{A}$ can be decomposed into a cascade $\mathcal{B}_1 \circ \cdots \circ \mathcal{B}_n$ where each component $\mathcal{B}_i$ is a two-state reset automaton

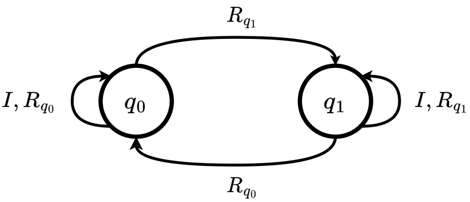
Two-states reset automaton



A transition labelled with σ can be of two types:

- Identity transition: $\forall q \;\; \delta(q, \sigma) = q$
- Reset transition on state q: $\forall q' \;\; \delta(q', \sigma) = q$

Two-states reset automaton



A transition labelled with σ can be of two types:

- Identity transition: $\forall q \;\; \delta(q, \sigma) = q$
- Reset transition on state q: $\forall q' \;\; \delta(q', \sigma) = q$

The alphabet Σ can be partitioned into sets of symbols I, R_{q_0}, R_{q_1} which respectively induce identity transitions, reset transitions entering state q_0 and reset transitions entering state q_1. We denote with R the set of symbols which induce a reset transition on some state q.

B-RASP state simulation

Let $\mathcal{A}$ be an automaton and let $w = w_1 \dots w_n$ be an input word. Let τ be the run $s_0, \dots, s_n$ induced by w on $\mathcal{A}$ starting from a certain state s.

B-RASP state simulation

Let $\mathcal{A}$ be an automaton and let $w = w_1 \ldots w_n$ be an input word. Let τ be the run $s_0, \ldots, s_n$ induced by w on $\mathcal{A}$ starting from a certain state s.
By definition of the run we have $s_{i+1} = \delta(s_i, w_i)$. s_i is the state reached before reading the i-th symbol.

B-RASP state simulation

Let $\mathcal{A}$ be an automaton and let $w = w_1 \dots w_n$ be an input word. Let τ be the run $s_0, \dots, s_n$ induced by w on $\mathcal{A}$ starting from a certain state s. By definition of the run we have $s_{i+1} = \delta(s_i, w_i)$. s_i is the state reached before reading the i-th symbol.

Definition

A B-RASP program p is said to *simulate* $\mathcal{A}$ starting from state s iff

- for each state q there is a boolean vector P_q;
- for every input word w, $P_q(i) = 1$ iff $s_i = q$.

B-RASP from state simulation to language recognition

Given a program p which simulates the automaton $\mathcal{A}$, for every state r we define the predicate that decides whether $\mathcal{A}$ started in state s ends up in state r after reading the symbol at position i:

$$A_r(i) \coloneqq \bigvee_{\substack{q \in Q \\ \sigma \in \Sigma \\ \delta(q,\sigma) = r}} P_q(i) \wedge Q_\sigma(i)$$

B-RASP from state simulation to language recognition

Given a program p which simulates the automaton $\mathcal{A}$, for every state r we define the predicate that decides whether $\mathcal{A}$ started in state s ends up in state r after reading the symbol at position i:

$$A_r(i) \coloneqq \bigvee_{\substack{q \in Q \\ \sigma \in \Sigma \\ \delta(q,\sigma)=r}} P_q(i) \wedge Q_\sigma(i)$$

Given a set of final states F we can define the output vector of the program as

$$Y \coloneqq \bigvee_{r \in F} A_r$$

and so $w \in \mathcal{L}(p)$ iff $w \in \mathcal{L}(\mathcal{A})$

Translating cascades to B-RASP

Lemma

Every cascade $\mathcal{C} = \mathcal{B}_1 \circ \cdots \circ \mathcal{B}_k$ of two-states reset automata can be simulated by a B-RASP program $p_{\mathcal{C}}$ from state $(s_1, \ldots, s_k)$.

Translating cascades to B-RASP

Lemma

Every cascade $\mathcal{C} = \mathcal{B}_1 \circ \cdots \circ \mathcal{B}_k$ of two-states reset automata can be simulated by a B-RASP program $p_{\mathcal{C}}$ from state $(s_1, \ldots, s_k)$.

Proof

We will proceed by induction on k the height of the cascade. The base case is a two-states reset automaton. For the inductive case, assume that the automaton $\mathcal{A} = \mathcal{B}_1 \circ \cdots \circ \mathcal{B}_k$ can be simulated by the program $p_{\mathcal{A}}$ and show that $\mathcal{C} = \mathcal{A} \circ \mathcal{B}_{k+1}$ can be simulated by the program $p_{\mathcal{C}}$.

Translating cascades to B-RASP: base case

Lemma

Every cascade $\mathcal{C} = \mathcal{B}_1 \circ \cdots \circ \mathcal{B}_k$ of two-states reset automata can be simulated by a B-RASP program $p_{\mathcal{C}}$ from state $(s_1, \ldots, s_k)$.

Proof (cont.)

Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a reset automaton. The following program $p_{\mathcal{A}}$, simulates $\mathcal{A}$ starting from state $s \in Q$. Q_σ denotes the input vector of σ.

$$P_q(i) \coloneqq \max_j \left[j < i, \bigvee_{\sigma \in R} Q_\sigma(j) \right] \bigvee_{\sigma \in R_q} Q_\sigma(j) : 0 \qquad \text{if } q \neq s$$

$$P_q(i) \coloneqq \max_j \left[j < i, \bigvee_{\sigma \in R} Q_\sigma(j) \right] \bigvee_{\sigma \in R_q} Q_\sigma(j) : 1 \qquad \text{if } q = s$$

Translating cascades to B-RASP: inductive case

Lemma

Every cascade $\mathcal{C} = \mathcal{B}_1 \circ \cdots \circ \mathcal{B}_k$ of two-states reset automata can be simulated by a B-RASP program $p_{\mathcal{C}}$ from state $(s_1, \ldots, s_k)$.

Proof (cont.)

The automaton $\mathcal{A} = (\Sigma, Q_{\mathcal{A}}, \delta_{\mathcal{A}})$ is simulated by $p_{\mathcal{A}}$. $P_{q_{\mathcal{A}}}$ is the predicate for state $q_{\mathcal{A}}$.
The reset automaton $\mathcal{B} = (\Sigma \times Q, Q_{\mathcal{B}}, \delta_{\mathcal{B}})$ is simulated by $p_{\mathcal{B}}$. $P_{q_{\mathcal{B}}}$ is the predicate for state $q_{\mathcal{B}}$.
Define $Q'_{(q_{\mathcal{A}},\sigma)} = P_{q_{\mathcal{A}}} \wedge Q_\sigma$ and $P'_{q_{\mathcal{B}}}$ as a clone of $P_{q_{\mathcal{B}}}$ where $Q_{(q_{\mathcal{A}},\sigma)}$ is replaced by $Q'_{(q_{\mathcal{A}},\sigma)}$.
For a state $(s_{\mathcal{A}}, s_{\mathcal{B}}) \in Q_{\mathcal{A}} \times Q_{\mathcal{B}}$ of $\mathcal{C} = \mathcal{A} \circ \mathcal{B}$ we define the predicate $P_{(s_{\mathcal{A}},s_{\mathcal{B}})}$ as $P_{(s_{\mathcal{A}},s_{\mathcal{B}})} \coloneqq P_{s_{\mathcal{A}}} \wedge P'_{s_{\mathcal{B}}}$ □

Conclusions

Further results

Well known results of LTL have deep consequences on transformers

Lemma

Pure Past LTL (pLTL) is equivalent to LTL

Lemma

LTL with non-strict operators recognize exactly the stutter-invariant star-free languages

Lemma

There exists a language L_k such that no LTL formula with temporal depth $2k$ recognizes L_k, but there exists an LTL formula with temporal depth $2k+1$ which recognizes L_k

Further results

Well known results of LTL have deep consequences on transformers

Lemma

Transformers with only future-masked rightmost-hard attention recognize exactly the star-free languages.

Lemma

LTL with non-strict operators recognize exactly the stutter-invariant star-free languages

Lemma

There exists a language L_k such that no LTL formula with temporal depth $2k$ recognizes L_k, but there exists an LTL formula with temporal depth $2k+1$ which recognizes L_k

Further results

Well known results of LTL have deep consequences on transformers

Lemma

Transformers with only future-masked rightmost-hard attention recognize exactly the star-free languages.

Lemma

Masked hard-attention transformers with only non-strict masking recognize exactly the stutter-invariant star-free languages.

Lemma

There exists a language L_k such that no LTL formula with temporal depth $2k$ recognizes L_k, but there exists an LTL formula with temporal depth $2k+1$ which recognizes L_k

Further results

Well known results of LTL have deep consequences on transformers

Lemma

Transformers with only future-masked rightmost-hard attention recognize exactly the star-free languages.

Lemma

Masked hard-attention transformers with only non-strict masking recognize exactly the stutter-invariant star-free languages.

Lemma

There exists a language L_k such that no multi-head masked hard-attention transformer of depth k recognizes L_k, but there exists a transformer of depth $O(k)$ which recognizes L_k

Depth Hierarchy
This last result is in contrast to the universal approximation theorem for standard FFN

equences on transformers

most-hard attention recognize

Masked hard-attention transformers with only non-strict masking recognize exactly the stutter-invariant star-free languages.

Lemma

There exists a language L_k such that no multi-head masked hard-attention transformer of depth k recognizes L_k, but there exists a transformer of depth $O(k)$ which recognizes L_k

Conclusions and final comments

- **Recap**
  - B-RASP has been used as an intermediate language which facilitates translation to LTL
  - Both SA and FF sub-layers of a transformer play a significant role. FF corresponds to boolean operators while SA corresponds to temporal operators
- **Future Work**
  - Extend the study to soft-attention transformers
  - Extend the study to encoder-decoder transformers
- **My Questions**
  - How does training transformers compare to automata learning algorithms?
  - LTL is used to derive results about transformers: is the converse possible?
  - How do reset automata cascades connect to transformer expressiveness?

Bibliography

Main paper

Andy Yang, David Chiang, and Dana Angluin. *Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages*. 2024. URL: https://arxiv.org/abs/2310.13897

LTL reference

Geatti Luca. *Temporal Logic Specifications: Expressiveness, Satisfiability And Realizability*. 2022

Automata Cascades reference

Borelli Roberto, Geatti Luca, Montali Marco, and Montanari Angelo. *On Cascades of Reset Automata*. 2025