Non-Emptiness for UHATs Is $\mathrm{EXPSPACE}$-Hard

Hao Su

July 3, 2026

Problem Statement & Proof Roadmap

Non-Emptiness for UHATs Is EXPSPACE-hard

Problem (Non-Emptiness for UHAT)

Given a UHAT T, decide if $L(T) \neq \emptyset$.

[1] Bergstraßer, Cotterell, and Lin, "Transformers Are Inherently Succinct," ICLR 2026.

Non-Emptiness for UHATs Is EXPSPACE-hard

Problem (Non-Emptiness for UHAT)

Given a UHAT T, decide if $L(T) \neq \emptyset$.

Why is this interesting?

Non-emptiness is a minimal verification question. If even this is EXPSPACE-hard, then richer verification tasks such as equivalence, universality, or property checking should not be expected to be easy in full generality.

[1] Bergstraßer, Cotterell, and Lin, "Transformers Are Inherently Succinct," ICLR 2026.

Non-Emptiness for UHATs Is EXPSPACE-hard

Problem (Non-Emptiness for UHAT)

Given a UHAT T, decide if $L(T) \neq \emptyset$.

Why is this interesting?

Non-emptiness is a minimal verification question. If even this is EXPSPACE-hard, then richer verification tasks such as equivalence, universality, or property checking should not be expected to be easy in full generality.

Today we show one of the main results by Bergstraßer et al.[1]:

Theorem

The non-emptiness problem for UHAT is EXPSPACE-hard.

[1] Bergstraßer, Cotterell, and Lin, "Transformers Are Inherently Succinct," ICLR 2026.

Proof Roadmap

Theorem (2^n-tiling)

2^n-tiling problem (will be introduced) is EXPSPACE*-complete*[2].

[2] F. Schwarzentruber, "The Complexity of Tiling Problems," arXiv:1907.00102, 2019.

Proof Roadmap

Theorem (2^n-tiling)

2^n-tiling problem (will be introduced) is $\mathrm{EXPSPACE}$-complete[2].

Lemma (2^n-tiling $\rightarrow$ B-RASP programs)

2^n-tilling problems are polynomial time reducible to the non-emptiness problems for B-RASP programs.

[2] F. Schwarzentruber, "The Complexity of Tiling Problems," arXiv:1907.00102, 2019.

Proof Roadmap

Theorem (2^n-tiling)

2^n-tiling problem (will be introduced) is $\mathrm{EXPSPACE}$-complete[2].

Lemma (2^n-tiling $\rightarrow$ B-RASP programs)

2^n-tilling problems are polynomial time reducible to the non-emptiness problems for B-RASP programs.

Lemma (B-RASP programs $\rightarrow$ UHATs)

Non-emptiness problems for UHATs are polynomial time reducible to the non-emptiness problems for B-RASP programs.

[2] F. Schwarzentruber, "The Complexity of Tiling Problems," arXiv:1907.00102, 2019.

Proof Roadmap

Theorem (2^n-tiling)

2^n-tiling problem (will be introduced) is $\mathrm{EXPSPACE}$*-complete*[2].

Lemma (2^n-tiling $\rightarrow$ B-RASP programs)

2^n-tilling problems are polynomial time reducible to the non-emptiness problems for B-RASP programs.

Lemma (B-RASP programs $\rightarrow$ UHATs)

Non-emptiness problems for UHATs are polynomial time reducible to the non-emptiness problems for B-RASP programs.

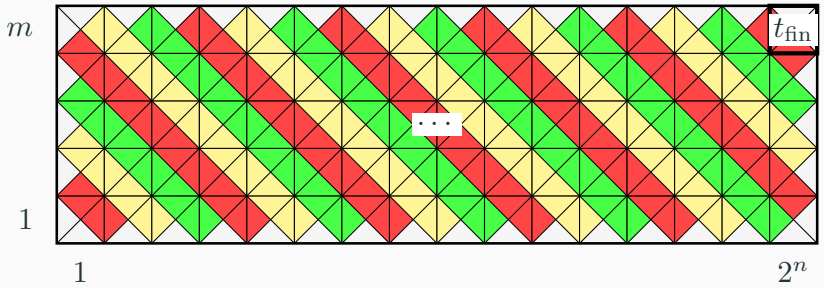
Theorem (A Lower Bound)

The non-emptiness problem for UHAT is $\mathrm{EXPSPACE}$*-hard.*

$$2^n\text{-Tiling} \leq_p \text{B-RASP Non-Emptiness} \leq_p \text{UHAT Non-Emptiness}$$

[2] F. Schwarzentruber, "The Complexity of Tiling Problems," arXiv:1907.00102, 2019.

2^n-tiling Problem (EXPSPACE-complete!)

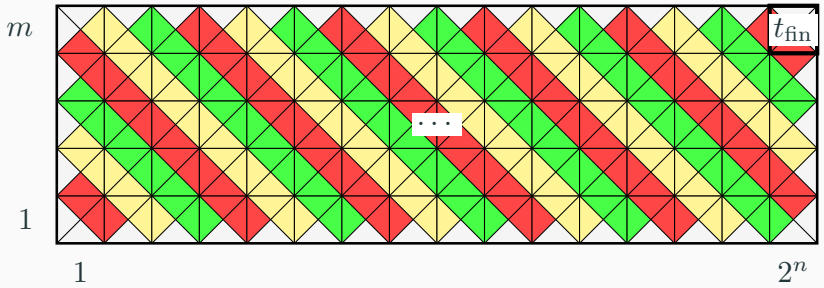


Definition (Tiles)

A *tile* is a quadruple $t \in \mathbb{N}_0^4$, where we write

$$t = \langle \text{left}(t), \text{up}(t), \text{right}(t), \text{down}(t) \rangle.$$

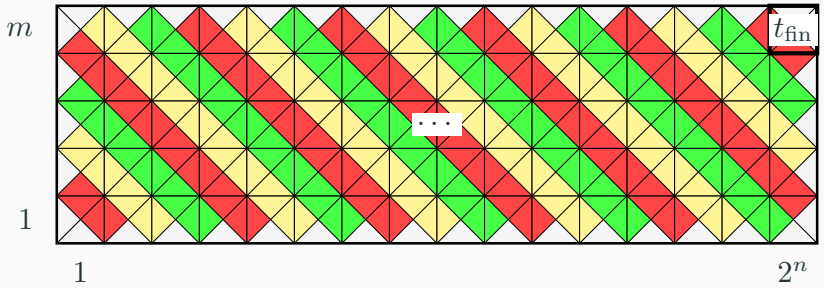
2^n-tiling Problem (EXPSPACE-complete!)



Problem (2^n-tiling)

Given (1) An integer $n > 0$ in unary, (2) a finite set T of tiles, and (3) a tile $t_{fin} \in T$,

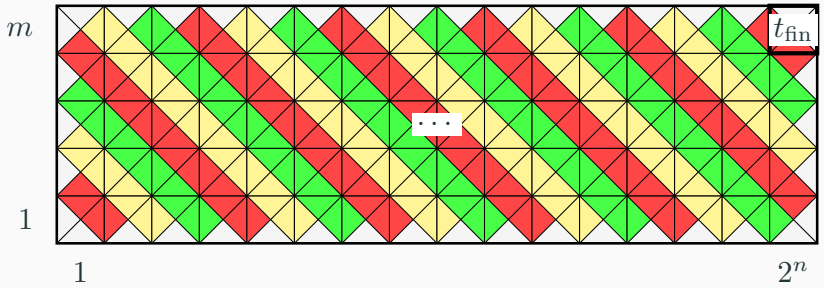
2^n-tiling Problem (EXPSPACE-complete!)



Problem (2^n-tiling)

Given (1) An integer $n > 0$ in unary, (2) a finite set T of tiles, and (3) a tile $t_{fin} \in T$, does there exist an interger $m > 0$ and a function $\tau : \{1, \ldots, 2^n\} \times \{1, \ldots, m\} \to T$

2^n-tiling Problem (EXPSPACE-complete!)



Problem (2^n-tiling)

Given (1) An integer $n > 0$ in unary, (2) a finite set T of tiles, and (3) a tile $t_{fin} \in T$, does there exist an interger $m > 0$ and a function $\tau : \{1, \ldots, 2^n\} \times \{1, \ldots, m\} \to T$ such that (1) $\tau(2^n, m) = t_{fin}$, (2) $\mathrm{down}(\tau(i, 1)) = \mathrm{up}(\tau(i, m)) = 0$ for all $1 \leq i \leq 2^n$, (3) $\mathrm{left}(\tau(1, j)) = \mathrm{right}(\tau(2^n, j)) = 0$ for all $1 \leq j \leq m$, (4) $\mathrm{right}(\tau(i, j)) = \mathrm{left}(\tau(i + 1, j))$ for all $1 \leq i < 2^n$ and $1 \leq j \leq m$, and (5) $\mathrm{up}(\tau(i, j)) = \mathrm{down}(\tau(i, j + 1))$ for all $1 \leq i \leq 2^n$ and $1 \leq j < m$.

2^n-tiling $\leq_p$ B-RASP non-emptiness

Recap on B-RASP programs

Vectors P_i in Restricted Access Sequence Processing (RASP)

- A feature vector assigns one value to each position of the input.
- A Boolean vector can record whether some property is true at each position.
- $Q_a(i) = 1 \Leftrightarrow w(i) = a$.

	1	2	3	4	5
input w	a	b	b	a	c
$P_1(Q_a)$	1	0	0	1	0
$P_2(Q_b)$	0	1	1	0	0
$P_3(Q_c)$	0	0	0	0	1
P_4	0	1	0	0	0

P_4: position i is the first b.

Recap on B-RASP programs

Vectors P_i in Restricted Access Sequence Processing (RASP)

- A feature vector assigns one value to each position of the input.
- A Boolean vector can record whether some property is true at each position.
- $Q_a(i) = 1 \Leftrightarrow w(i) = a$.

Definition (Sketching B-RASP)

- A *B-RASP* program *builds* new features with basic token features $(Q_a, Q_b, \dots)$.
- One of the vectors is designated as the *output vector* deciding acceptance.

	1	2	3	4	5
input w	a	b	b	a	c
$P_1(Q_a)$	1	0	0	1	0
$P_2(Q_b)$	0	1	1	0	0
$P_3(Q_c)$	0	0	0	0	1
P_4	0	1	0	0	0

P_4: position i is the first b.

Recap on B-RASP programs: Building Operations

Position-wise operation

$P_{t+1}(i) := R(i)$, where $R(i)$ is some Boolean combination of $\{P_1(i), \ldots, P_t(i)\}$.

Recap on B-RASP programs: Building Operations

Position-wise operation

$P_{t+1}(i) := R(i)$, where $R(i)$ is some Boolean combination of $\{P_1(i), \ldots, P_t(i)\}$.

Attention operation

An attention operation defines a new feature vector P_{t+1} by letting each position i look at one selected position j,

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

where S, V, D are Boolean combinations of $\{P_1(i), \ldots, P_t(i)\}$.

- $M(i,j)$ is the mask, saying which positions j are visible from i.
- $S(i,j)$ is the score predicate, saying which visible positions are relevant.
- $\triangleright_j$ means: choose the rightmost visible j such that $S(i,j) = 1$.
- If such a position j exists, set $P_{t+1}(i) := V(i,j)$. If no such position exists, use the default value: $P_{t+1}(i) := D(i)$.

2^n-tiling $\leq_p$ B-RASP non-emptiness: Encoding τ as a word

We encode a configuration of tiles (τ) as a word over the alphabet $\Sigma := T \cup \{0, 1, \#\}$.

2^n-tiling $\leq_p$ B-RASP non-emptiness: Encoding τ as a word

We encode a configuration of tiles (τ) as a word over the alphabet $\Sigma := T \cup \{0, 1, \#\}$.

Encoding Scheme

1. Define $\mathrm{enc}_\tau : \{1, \ldots, 2^n\} \times \{1, \ldots, m\} \to \Sigma^*$ by $\mathrm{enc}_\tau(i, j) := \langle i - 1 \rangle \tau(i, j) \#$ for all $i \in [1, 2^n]$ and $j \in [1, m]$, where $\langle i - 1 \rangle$ denotes the binary encoding of $i - 1$.

2^n-tiling $\leq_p$ B-RASP non-emptiness: Encoding τ as a word

We encode a configuration of tiles (τ) as a word over the alphabet $\Sigma := T \cup \{0, 1, \#\}$.

Encoding Scheme

1. Define $\mathrm{enc}_\tau : \{1, \ldots, 2^n\} \times \{1, \ldots, m\} \to \Sigma^*$ by $\mathrm{enc}_\tau(i, j) := \langle i - 1 \rangle \tau(i, j) \#$ for all $i \in [1, 2^n]$ and $j \in [1, m]$, where $\langle i - 1 \rangle$ denotes the binary encoding of $i - 1$.
2. Define $\mathrm{enc}(\tau) := \mathrm{enc}_\tau(1, 1) \cdots \mathrm{enc}_\tau(2^n, 1) \mathrm{enc}_\tau(1, 2) \cdots \mathrm{enc}_\tau(2^n, m)$.

Example

$0000 a_1 \# 0001 a_2 \# 0010 a_3 \# \cdots \# 1111 a_{2^4} \#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

2^n-tiling $\leq_p$ B-RASP non-emptiness: Encoding τ as a word

We encode a configuration of tiles (τ) as a word over the alphabet $\Sigma := T \cup \{0, 1, \#\}$.

Encoding Scheme

1. Define $\mathrm{enc}_\tau : \{1, \ldots, 2^n\} \times \{1, \ldots, m\} \to \Sigma^*$ by $\mathrm{enc}_\tau(i, j) := \langle i - 1 \rangle \tau(i, j) \#$ for all $i \in [1, 2^n]$ and $j \in [1, m]$, where $\langle i - 1 \rangle$ denotes the binary encoding of $i - 1$.
2. Define $\mathrm{enc}(\tau) := \mathrm{enc}_\tau(1, 1) \cdots \mathrm{enc}_\tau(2^n, 1) \mathrm{enc}_\tau(1, 2) \cdots \mathrm{enc}_\tau(2^n, m)$.

Example

$0000 a_1 \# 0001 a_2 \# 0010 a_3 \# \cdots \# 1111 a_{2^4} \#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We construct a B-RASP program that accepts $\mathrm{enc}(\tau)$ iff τ satisfies the conditions.

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

$$A_T(i) := \blacktriangleright_j\, [j < i, 1] \bigvee_{t \in T} Q_t(j) : 0$$

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

$$A_T(i) :=\blacktriangleright_j\ [j < i, 1] \bigvee_{t \in T} Q_t(j) : 0$$

$$A_{C,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_0(j) \vee Q_1(j) : 0$$

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

$$A_T(i) :=\blacktriangleright_j\ [j < i, 1] \bigvee_{t \in T} Q_t(j) : 0$$

$$A_{C,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_0(j) \vee Q_1(j) : 0$$

$$A_{C,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{C,k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

$$A_T(i) :=\blacktriangleright_j\ [j < i, 1] \bigvee_{t \in T} Q_t(j) : 0$$

$$A_{C,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_0(j) \vee Q_1(j) : 0$$

$$A_{C,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{C,k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

$$A_{\#,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_{\#}(j) : 1$$

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

$$A_T(i) :=\blacktriangleright_j\ [j < i, 1] \bigvee_{t \in T} Q_t(j) : 0$$

$$A_{C,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_0(j) \vee Q_1(j) : 0$$

$$A_{C,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{C,k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

$$A_{\#,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_{\#}(j) : 1$$

$$A_{\#,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{\#,k-1}(j) : 1 \qquad \text{for } k = 2, \ldots, n+1$$

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

$$A_T(i) :=\blacktriangleright_j\ [j < i, 1] \bigvee_{t\in T} Q_t(j) : 0$$

$$A_{C,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_0(j) \vee Q_1(j) : 0$$

$$A_{C,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{C,k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

$$A_{\#,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_\#(j) : 1$$

$$A_{\#,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{\#,k-1}(j) : 1 \qquad \text{for } k = 2, \ldots, n+1$$

$$A_{enc}(i) := (Q_\#(i) \to A_T(i)) \wedge \left(\left(\bigvee_{t\in T} Q_t(i)\right) \to \left(\bigwedge_{k=1}^{n} A_{C,k}(i)\right) \wedge A_{\#,n+1}(i)\right)$$

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

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

The construction: form checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

We first check if the input is of the form $(\{0,1\}^n T\#)^*$ by the following vectors.

$$A_T(i) :=\blacktriangleright_j\ [j < i, 1] \bigvee_{t\in T} Q_t(j) : 0$$

$$A_{C,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_0(j) \vee Q_1(j) : 0$$

$$A_{C,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{C,k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

$$A_{\#,1}(i) :=\blacktriangleright_j\ [j < i, 1] Q_\#(j) : 1$$

$$A_{\#,k}(i) :=\blacktriangleright_j\ [j < i, 1] A_{\#,k-1}(j) : 1 \qquad \text{for } k = 2, \ldots, n+1$$

$$A_{enc}(i) := (Q_\#(i) \to A_T(i)) \wedge \left(\left(\bigvee_{t\in T} Q_t(i)\right) \to \left(\bigwedge_{k=1}^{n} A_{C,k}(i)\right) \wedge A_{\#,n+1}(i)\right)$$

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

Require $A(|enc(\tau)|) = 1$.

The construction: counter checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

The construction: counter checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

$$C_1(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]Q_1(j) : 0$$

The construction: counter checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

$$C_1(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]Q_1(j) : 0$$
$$C_k(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]C_{k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

The construction: counter checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

$$C_1(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]Q_1(j) : 0$$

$$C_k(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]C_{k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

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

$$\left(\bigwedge_{r=k+1}^{n} C_r(i) \leftrightarrow C_r(j) \right) : 0$$

The construction: counter checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

$$C_1(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]Q_1(j) : 0$$

$$C_k(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]C_{k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

$$C_{+1}(i) :=\blacktriangleright_j\ [j < i, Q_\#(j)] \bigvee_{k=1}^{n} \left(\bigwedge_{r=1}^{k-1} \neg C_r(i) \wedge C_r(j) \right) \wedge C_k(i) \wedge \neg C_k(j) \wedge \left(\bigwedge_{r=k+1}^{n} C_r(i) \leftrightarrow C_r(j) \right) : 0$$

$$C_{1\to 0}(i) :=\blacktriangleright_j\ [j < i, Q_\#(j)] \bigwedge_{k=1}^{n} \neg C_k(i) \wedge C_k(j) : \bigwedge_{k=1}^{n} \neg C_k(i)$$

The construction: counter checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

$$C_1(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]Q_1(j) : 0$$

$$C_k(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]C_{k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

$$C_{+1}(i) :=\blacktriangleright_j\ [j < i, Q_\#(j)] \bigvee_{k=1}^{n} \left(\bigwedge_{r=1}^{k-1} \neg C_r(i) \wedge C_r(j) \right) \wedge C_k(i) \wedge \neg C_k(j) \wedge \left(\bigwedge_{r=k+1}^{n} C_r(i) \leftrightarrow C_r(j) \right) : 0$$

$$C_{1\to 0}(i) :=\blacktriangleright_j\ [j < i, Q_\#(j)] \bigwedge_{k=1}^{n} \neg C_k(i) \wedge C_k(j) : \bigwedge_{k=1}^{n} \neg C_k(i)$$

$$C(i) :=\blacktriangleright_j\ [j < i, Q_\#(j) \wedge \neg C_{1\to 0}(j) \wedge \neg C_{+1}(j)]0 : C_{1\to 0}(i) \vee C_{+1}(i).$$

The construction: counter checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

$$C_1(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]Q_1(j) : 0$$

$$C_k(i) :=\blacktriangleright_j\ [j < i, Q_0(j) \vee Q_1(j)]C_{k-1}(j) : 0 \qquad \text{for } k = 2, \ldots, n$$

$$C_{+1}(i) :=\blacktriangleright_j\ [j < i, Q_\#(j)] \bigvee_{k=1}^{n} \left(\bigwedge_{r=1}^{k-1} \neg C_r(i) \wedge C_r(j) \right) \wedge C_k(i) \wedge \neg C_k(j) \wedge \left(\bigwedge_{r=k+1}^{n} C_r(i) \leftrightarrow C_r(j) \right) : 0$$

$$C_{1\to 0}(i) :=\blacktriangleright_j\ [j < i, Q_\#(j)] \bigwedge_{k=1}^{n} \neg C_k(i) \wedge C_k(j) : \bigwedge_{k=1}^{n} \neg C_k(i)$$

$$C(i) :=\blacktriangleright_j\ [j < i, Q_\#(j) \wedge \neg C_{1\to 0}(j) \wedge \neg C_{+1}(j)]0 : C_{1\to 0}(i) \vee C_{+1}(i).$$

Require $C(|enc(\tau)|) = 1$.

The construction: final tile checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

The construction: final tile checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

The construction: final tile checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

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

The construction: final tile checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

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

Require $F(|enc(\tau)|) = 1$.

The construction: boundary checking (1)

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

The construction: boundary checking (1)

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

The construction: boundary checking (1)

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

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

The construction: boundary checking (2)

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

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

The construction: boundary checking (2)

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

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

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

The construction: boundary checking (2)

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

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

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

Require $E(|enc(\tau)|) = 1$.

The construction: adjacency checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

The construction: adjacency checking

Example

$0000a_1\#0001a_2\#0010a_3\#\cdots\#1111a_{2^4}\#$. ($n = 4$, $m = 1$ and $a_i \in T$ for $i \in [2^4]$.)

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

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

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

Require $M(|enc(\tau)|) = 1$.

The construction: wrap up

Definition (Accpetance of the B-RASP program)

1. The output vector is defined by the conjunction

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

2. The B-RASP program accepts iff $Y(\ell) = 1$.

The construction: wrap up

Definition (Accpetance of the B-RASP program)

1. The output vector is defined by the conjunction

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

2. The B-RASP program accepts iff $Y(\ell) = 1$.

Observations

- The B-RASP program recognizes the set of all $enc(\tau)$ where τ satisfies the conditions above.
- The language recognized by the B-RASP program is non-empty iff the 2^n-tiling problem has a solution.

The construction: wrap up

Definition (Accpetance of the B-RASP program)

1. The output vector is defined by the conjunction

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

2. The B-RASP program accepts iff $Y(\ell) = 1$.

Observations

- The B-RASP program recognizes the set of all $enc(\tau)$ where τ satisfies the conditions above.
- The language recognized by the B-RASP program is non-empty iff the 2^n-tiling problem has a solution.
- The construction takes time polynomial in n.
- Why does this work?

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.Each attention operation acts like a small RAM query over the input sequence.

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.Each attention operation acts like a small RAM query over the input sequence.The program requires polynomially many such attention-based lookup features to verify encodings whose natural automaton representation would be exponentially larger.

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.Each attention operation acts like a small RAM query over the input sequence.The program requires polynomially many such attention-based lookup features to verify encodings whose natural automaton representation would be exponentially larger.

Disk analogy: Can apply polynomial requirements on the disk addresses. The data on the disk would then need a DFA with exponential states to check.

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.Each attention operation acts like a small RAM query over the input sequence.The program requires polynomially many such attention-based lookup features to verify encodings whose natural automaton representation would be exponentially larger.
Disk analogy: Can apply polynomial requirements on the disk addresses. The data on the disk would then need a DFA with exponential states to check.

Hence transformers no better than certain formal descriptions. We are building a succinct verifier of the 2^n-tiling problem.

However in Real Transformers

1. The number of layers and heads are fixed after training.

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.Each attention operation acts like a small RAM query over the input sequence.The program requires polynomially many such attention-based lookup features to verify encodings whose natural automaton representation would be exponentially larger.
Disk analogy: Can apply polynomial requirements on the disk addresses. The data on the disk would then need a DFA with exponential states to check.

Hence transformers no better than certain formal descriptions. We are building a succinct verifier of the 2^n-tiling problem.

However in Real Transformers

1. The number of layers and heads are fixed after training.
2. Soft attention is approximate, not unique hard attention.

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.Each attention operation acts like a small RAM query over the input sequence.The program requires polynomially many such attention-based lookup features to verify encodings whose natural automaton representation would be exponentially larger.
Disk analogy: Can apply polynomial requirements on the disk addresses. The data on the disk would then need a DFA with exponential states to check.

Hence transformers no better than certain formal descriptions. We are building a succinct verifier of the 2^n-tiling problem.

However in Real Transformers

1. The number of layers and heads are fixed after training.
2. Soft attention is approximate, not unique hard attention.
3. Learned heads are not guaranteed to implement exact Boolean predicates or exact argmax equality tests.

Intuition & Comparison with real transformers

Intuition

The B-RASP construction uses attention as an addressable lookup mechanism.Each attention operation acts like a small RAM query over the input sequence.The program requires polynomially many such attention-based lookup features to verify encodings whose natural automaton representation would be exponentially larger.
Disk analogy: Can apply polynomial requirements on the disk addresses. The data on the disk would then need a DFA with exponential states to check.

Hence transformers no better than certain formal descriptions. We are building a succinct verifier of the 2^n-tiling problem.

However in Real Transformers

1. The number of layers and heads are fixed after training.
2. Soft attention is approximate, not unique hard attention.
3. Learned heads are not guaranteed to implement exact Boolean predicates or exact argmax equality tests.

B-RASP non-emptiness $\leq_p$ UHAT non-emptiness

Formal statement

Lemma (B-RASP non-emptiness $\leq_p$ UHAT non-emptiness)

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

- $|\Sigma| \leq t < m$,
- $\blacklozenge \in \{\blacktriangleleft, \blacktriangleright\}$,
- $S(j)$ *is a Boolean combination 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.

Recap on UHAT: Masked Unique Hard-Attention (UHA)

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7$

Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.

Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.

Recap on UHAT: Masked Unique Hard-Attention (UHA)

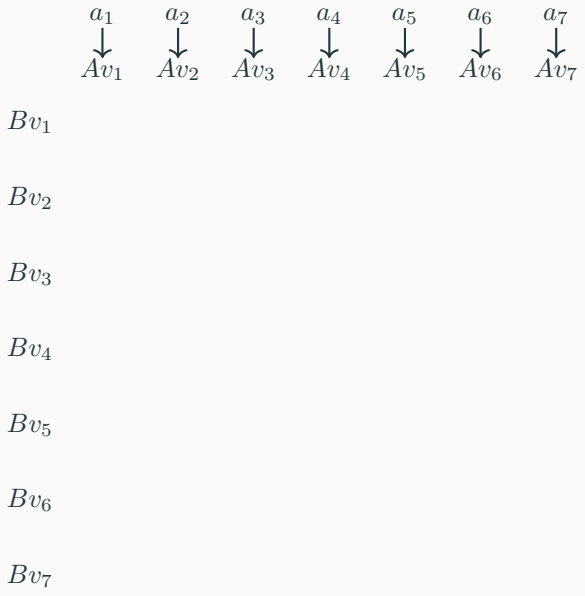
Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.

Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

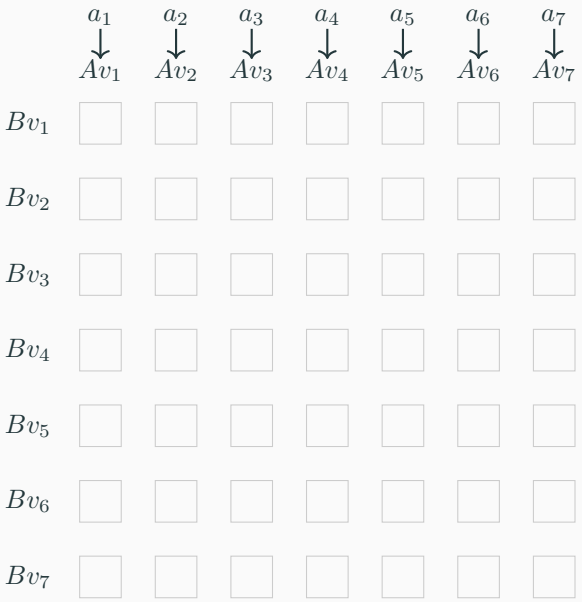
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.



Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

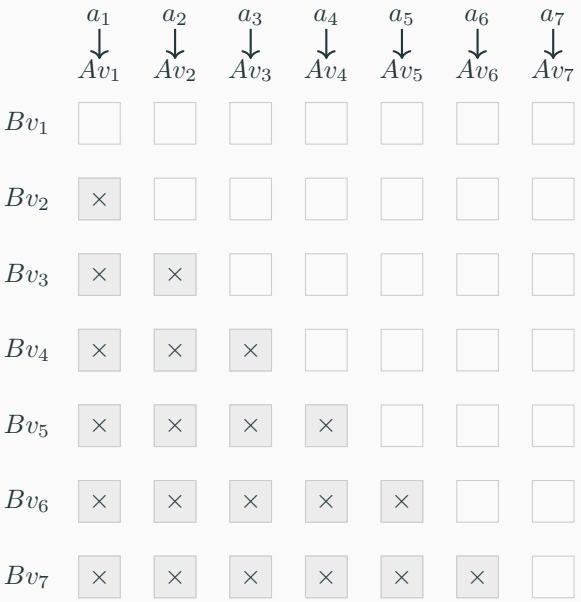
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.



Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

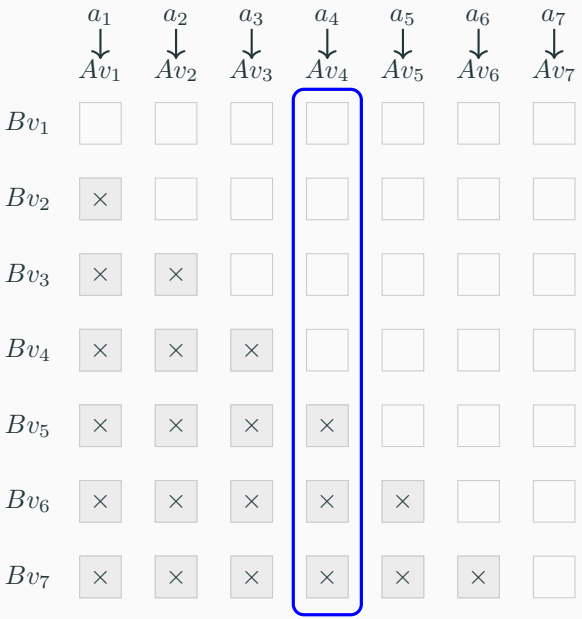
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.



Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

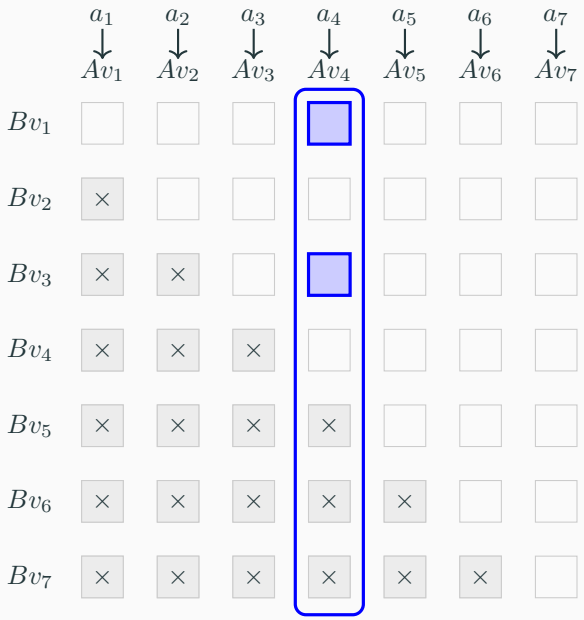
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.



Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

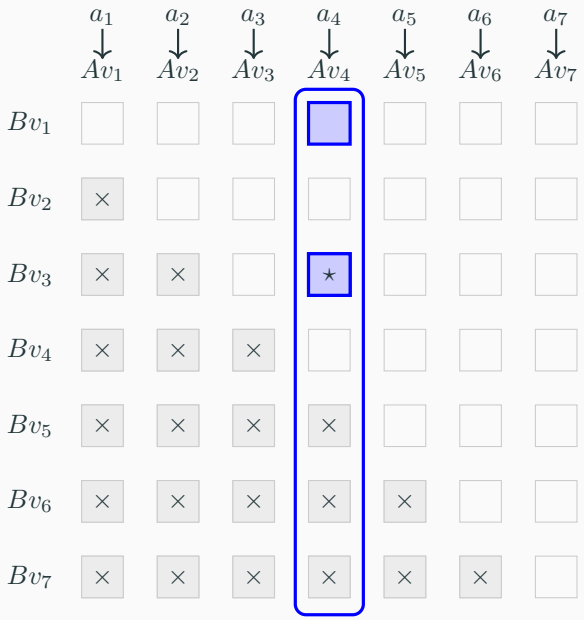
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.
- Score $S(v_i, v_j) = \langle A(v_i), B(v_j) \rangle$.



Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

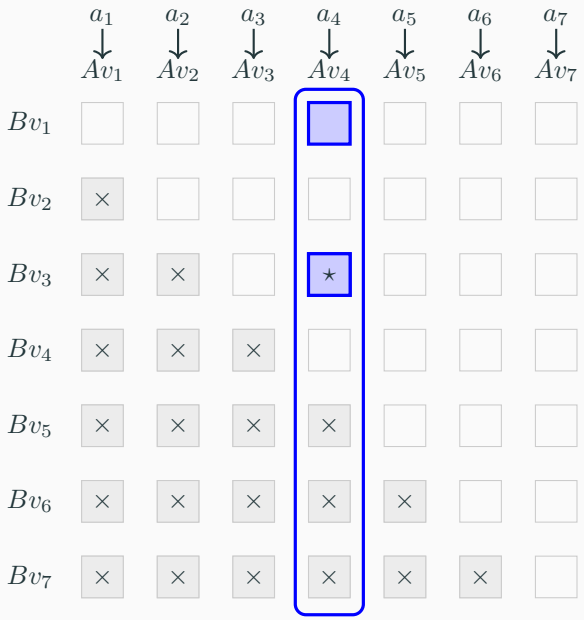
- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.
- Score $S(v_i, v_j) = \langle A(v_i), B(v_j) \rangle$.
- Tie Breaking τ. (why?)



Recap on UHAT: Masked Unique Hard-Attention (UHA)

Definition (Sketching UHA)

- Input sequence $a_1 \cdots a_n \in \Sigma^n$.
- $\mathrm{emb} : \Sigma \to \mathbb{Q}^r$.
- Query $A : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Key $B : \mathbb{Q}^r \to \mathbb{Q}^r$.
- Mask $M : \mathbb{N}^2 \to \{\top, \bot\}$.
- Score $S(v_i, v_j) = \langle A(v_i), B(v_j) \rangle$.
- Tie Breaking τ. (why?)
- Value Update $C : \mathbb{Q}^{2r} \to \mathbb{Q}^s$.
  For example, $v'_4 = C(v_3, v_4)$.



Recap on UHAT: ReLU, UHAT, and UHAT Acceptance

Definition (ReLU)

For each vector $v_i (i \in [n])$ and some k,

$$\mathrm{ReLU}(v_i) = (v_i[1, k-1], \max\{0, v_i(k)\}, v_i[k+1, n]).$$

Recap on UHAT: ReLU, UHAT, and UHAT Acceptance

Definition (ReLU)

For each vector $v_i (i \in [n])$ and some k,

$$\mathrm{ReLU}(v_i) = (v_i[1, k-1], \max\{0, v_i(k)\}, v_i[k+1, n]).$$

Definition (UHAT)

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

Recap on UHAT: ReLU, UHAT, and UHAT Acceptance

Definition (ReLU)

For each vector $v_i (i \in [n])$ and some k,

$$\mathrm{ReLU}(v_i) = (v_i[1, k-1], \max\{0, v_i(k)\}, v_i[k+1, n]).$$

Definition (UHAT)

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

Definition (UHAT Acceptance)

Assume that $\mathcal{T}$ is given together with an *acceptance vector* $t \in \mathbb{Q}^s$. The recognized language $L(\mathcal{T})$ is then the set of words on which $\mathcal{T}$ outputs a sequence $v_1, \ldots, v_n \in \mathbb{Q}^s$ with $\langle t, v_k \rangle > 0$, where $k \in [1, n]$ is either 1 or n depending on whether the first or last position is regarded the *output position* of $\mathcal{T}$.

B-RASP Program $\rightarrow$ UHAT: Embedding and Point-Wise Operation

Embedding

Each token $a \in \Sigma$ is embedded as a one hot vector according to Q_{a_i}'s.

	1	2	3	4	5
input w	a	b	b	a	c
$P_1(Q_a)$	1	0	0	1	0
$P_2(Q_b)$	0	1	1	0	0
$P_3(Q_c)$	0	0	0	0	1
P_4	0	1	0	0	0

P_4: position i is the first b.

B-RASP Program $\rightarrow$ UHAT: Embedding and Point-Wise Operation

Embedding

Each token $a \in \Sigma$ is embedded as a one hot vector according to Q_{a_i}'s.

Then we simulate the two operations: position-wise operation and attention operation.

	1	2	3	4	5
input w	a	b	b	a	c
$P_1(Q_a)$	1	0	0	1	0
$P_2(Q_b)$	0	1	1	0	0
$P_3(Q_c)$	0	0	0	0	1
P_4	0	1	0	0	0

P_4: position i is the first b.

B-RASP Program $\to$ UHAT: Embedding and Point-Wise Operation

Embedding

Each token $a \in \Sigma$ is embedded as a one hot vector according to Q_{a_i}'s.

Then we simulate the two operations: position-wise operation and attention operation.

Point-wise operation

- Formally, for some Boolean combination R, we set $P_{t+1}(i) := R(P_1(i), \ldots, P_t(i))$.
- In the B-RASP to UHAT translation, such Boolean combinations are simulated using affine transformations and ReLU layers.

	1	2	3	4	5
input w	a	b	b	a	c
$P_1(Q_a)$	1	0	0	1	0
$P_2(Q_b)$	0	1	1	0	0
$P_3(Q_c)$	0	0	0	0	1
P_4	0	1	0	0	0

P_4: position i is the first b.

Simulating position-wise B-RASP operations

Bottom-up simulation needing only $\neg$ and $\wedge$

The UHAT simulates R bottom-up using affine maps and ReLU layers:

$$\neg x = 1 - x,$$

Simulating position-wise B-RASP operations

Bottom-up simulation needing only $\neg$ and $\wedge$

The UHAT simulates R bottom-up using affine maps and ReLU layers:

$$\neg x = 1 - x,$$

and, for Boolean $x, y \in \{0, 1\}$,

$$x \wedge y = \max\{0, x + y - 1\}.$$

Simulating position-wise B-RASP operations

Bottom-up simulation needing only $\neg$ and $\wedge$

The UHAT simulates R bottom-up using affine maps and ReLU layers:

$$\neg x = 1 - x,$$

and, for Boolean $x, y \in \{0, 1\}$,

$$x \wedge y = \max\{0, x + y - 1\}.$$

Disjunction is obtained by De Morgan:

$$x \vee y = \neg(\neg x \wedge \neg y).$$

Simulating position-wise B-RASP operations

Bottom-up simulation needing only $\neg$ and $\wedge$

The UHAT simulates R bottom-up using affine maps and ReLU layers:

$$\neg x = 1 - x,$$

and, for Boolean $x, y \in \{0, 1\}$,

$$x \wedge y = \max\{0, x + y - 1\}.$$

Disjunction is obtained by De Morgan:

$$x \vee y = \neg(\neg x \wedge \neg y).$$

After computing the subformulas of R, the UHAT keeps only

$$(P_1(i), \ldots, P_t(i), P_{t+1}(i)).$$

Restricted B-RASP attention operations

Attention Operations

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

or the analogous leftmost version.
Here: $M(i,j)$ is the mask, $S(j)$ is a Boolean condition on the candidate source position j, and

$$\bigwedge_{k \in K} P_k(i) \leftrightarrow P_k(j)$$

requires selected Boolean features at i and j to match.

Restricted B-RASP attention operations

Attention Operations

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

or the analogous leftmost version.
Here: $M(i,j)$ is the mask, $S(j)$ is a Boolean condition on the candidate source position j, and

$$\bigwedge_{k \in K} P_k(i) \leftrightarrow P_k(j)$$

requires selected Boolean features at i and j to match.

Goal

Make UHAT hard attention select exactly the same j as the B-RASP operation.

Encoding the B-RASP score as a UHAT score

For each $k \in K$, the UHAT stores both

$$P_k(x) \qquad \text{and} \qquad 1 - P_k(x)$$

at every position x.
Use the attention score

$$\text{score}(i, j) = \sum_{k \in K} (P_k(i)P_k(j) + (1 - P_k(i))(1 - P_k(j))) - (1 - S(j)).$$

Encoding the B-RASP score as a UHAT score

For each $k \in K$, the UHAT stores both

$$P_k(x) \qquad \text{and} \qquad 1 - P_k(x)$$

at every position x.
Use the attention score

$$\text{score}(i, j) = \sum_{k \in K} (P_k(i)P_k(j) + (1 - P_k(i))(1 - P_k(j))) - (1 - S(j)).$$

For Boolean values,

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

Encoding the B-RASP score as a UHAT score

For each $k \in K$, the UHAT stores both

$$P_k(x) \qquad \text{and} \qquad 1 - P_k(x)$$

at every position x.
Use the attention score

$$\text{score}(i, j) = \sum_{k \in K} (P_k(i)P_k(j) + (1 - P_k(i))(1 - P_k(j))) - (1 - S(j)).$$

For Boolean values,

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

Hence

$$\text{score}(i, j) = |\{k \in K : P_k(i) = P_k(j)\}| - (1 - S(j)).$$

Encoding the B-RASP score as a UHAT score

For each $k \in K$, the UHAT stores both

$$P_k(x) \qquad \text{and} \qquad 1 - P_k(x)$$

at every position x.
Use the attention score

$$\text{score}(i, j) = \sum_{k \in K} (P_k(i)P_k(j) + (1 - P_k(i))(1 - P_k(j))) - (1 - S(j)).$$

For Boolean values,

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

Hence

$$\text{score}(i, j) = |\{k \in K : P_k(i) = P_k(j)\}| - (1 - S(j)).$$

The maximum possible score is $|K|$, attained exactly when

$$S(j) = 1 \quad \text{and} \quad P_k(i) = P_k(j) \text{ for all } k \in K.$$

Handling the default case

UHAT attention always selects some unmasked maximizer if one exists.

Handling the default case

UHAT attention always selects some unmasked maximizer if one exists. B-RASP attention is different: if no j satisfies the Boolean score predicate, it returns the default value $D(i)$.

Handling the default case

UHAT attention always selects some unmasked maximizer if one exists. B-RASP attention is different: if no j satisfies the Boolean score predicate, it returns the default value $D(i)$. So after UHAT attention selects a position j_i, the UHAT recomputes the validity flag

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

Handling the default case

UHAT attention always selects some unmasked maximizer if one exists. B-RASP attention is different: if no j satisfies the Boolean score predicate, it returns the default value $D(i)$. So after UHAT attention selects a position j_i, the UHAT recomputes the validity flag

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

Then it computes

$$P_{t+1}(i) = \big(R(i) \wedge V(i, j_i)\big) \vee \big(\neg R(i) \wedge D(i)\big).$$

Thus:

$$R(i) = 1 \quad \Rightarrow \quad P_{t+1}(i) = V(i, j_i),$$

while

$$R(i) = 0 \quad \Rightarrow \quad P_{t+1}(i) = D(i).$$

Handling the default case

UHAT attention always selects some unmasked maximizer if one exists. B-RASP attention is different: if no j satisfies the Boolean score predicate, it returns the default value $D(i)$. So after UHAT attention selects a position j_i, the UHAT recomputes the validity flag

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

Then it computes

$$P_{t+1}(i) = \big(R(i) \wedge V(i, j_i)\big) \vee \big(\neg R(i) \wedge D(i)\big).$$

Thus:

$$R(i) = 1 \quad \Rightarrow \quad P_{t+1}(i) = V(i, j_i),$$

while

$$R(i) = 0 \quad \Rightarrow \quad P_{t+1}(i) = D(i).$$

This exactly matches the B-RASP semantics.

Preserving acceptance

The UHAT simulation maintains the invariant:

$$\text{after simulating } P_1, \ldots, P_t, \quad \text{position } i \text{ stores } (P_1(i), \ldots, P_t(i)).$$

Preserving acceptance

The UHAT simulation maintains the invariant:

$$\text{after simulating } P_1, \ldots, P_t, \quad \text{position } i \text{ stores } (P_1(i), \ldots, P_t(i)).$$

Acceptance

If P_t is the B-RASP output vector, stop constructing, and use acceptance one-hot vector e_t and the same output position k as P_t.
Then the UHAT accepts iff

$$\langle e_t, (P_1(k), \ldots, P_t(k)) \rangle > 0.$$

Since all coordinates are Boolean, this is equivalent to $P_t(k) = 1$.

Preserving acceptance

The UHAT simulation maintains the invariant:

$$\text{after simulating } P_1, \ldots, P_t, \quad \text{position } i \text{ stores } (P_1(i), \ldots, P_t(i)).$$

Acceptance

If P_t is the B-RASP output vector, stop constructing, and use acceptance one-hot vector e_t and the same output position k as P_t.
Then the UHAT accepts iff

$$\langle e_t, (P_1(k), \ldots, P_t(k)) \rangle > 0.$$

Since all coordinates are Boolean, this is equivalent to $P_t(k) = 1$.

Therefore the UHAT recognizes exactly the same language as the B-RASP program.

Questions?