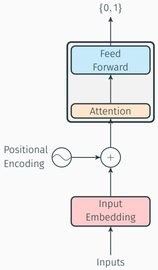
Masked Hard-Attention Transformers and Boolean RASP Recognize Exactly the Star-Free Languages

Dana Angluin, David Chiang, Andy Yang

The Big Picture: expressivity and logic



$$(\forall i)(Q_a)$$
$$(\forall i)(\forall j)(i < j \rightarrow Q_a(i) \wedge Q_b(j))$$
etc.

What languages are recognized by transformer encoders?

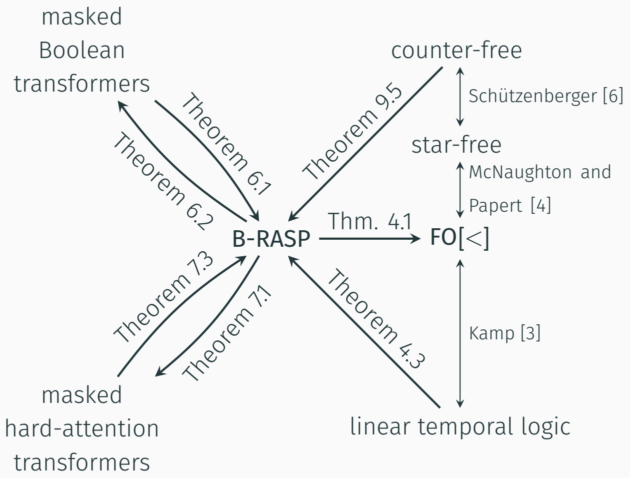
What languages are defined by logical formulas?

For a survey of papers in this area (including this one): Strobl et al. [7], "Transformers as Recognizers of Formal Languages: A Survey on Expressivity"

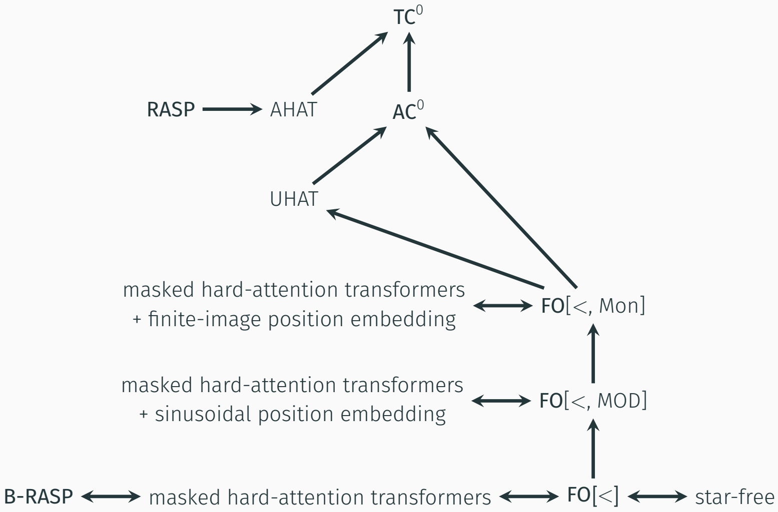
Questions to Consider

- Expressivity?
- Learnability?
- Interpretability?
- Improvements?

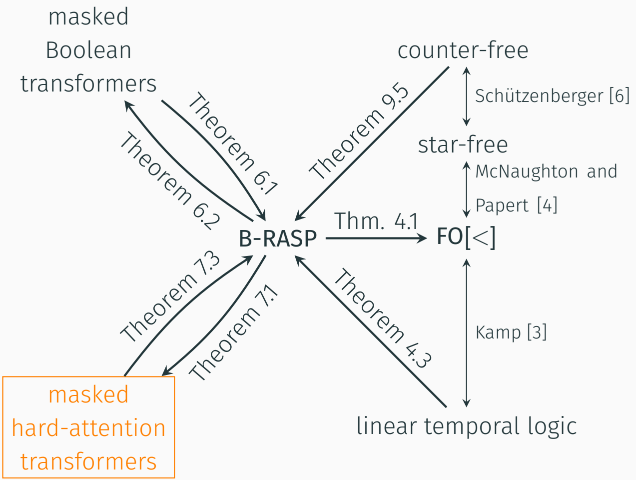
Our Results


masked Boolean transformers
counter-free
Schützenberger [6]
star-free
McNaughton and Papert [4]
Theorem 6.1
Theorem 6.2
Theorem 9.5
B-RASP
Thm. 4.1
FO[<]
Theorem 7.3
Theorem 7.1
Theorem 4.3
Kamp [3]
masked hard-attention transformers
linear temporal logic

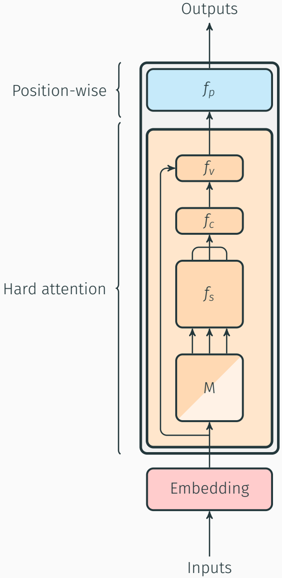
Contextualizing Our Results



Masked Hard-Attention Transformers



Masked Hard-Attention Transformers



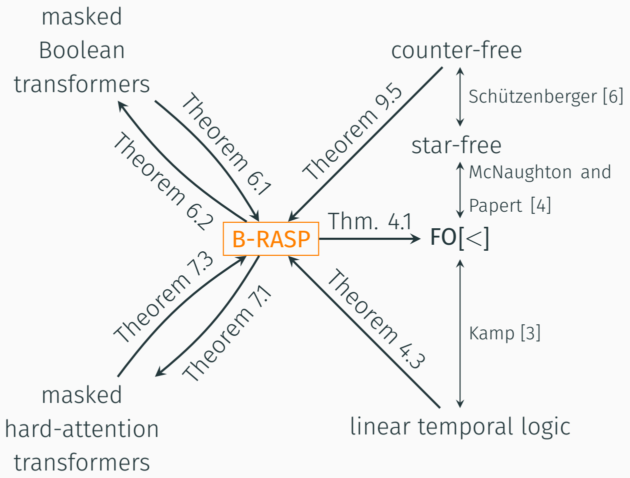
3-layer feed-forward network with ReLU activation

Values: any linear function with residual connection

Leftmost or rightmost maximum

Attention scores: any bilinear function

Attention masking: strict future, strict past, or none

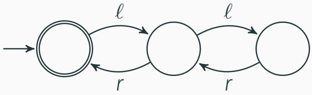

masked
Boolean
transformers
counter-free
Schützenberger [6]
star-free
McNaughton and
Papert [4]
Theorem 6.1
Theorem 6.2
Theorem 9.5
B-RASP
Thm. 4.1
FO[<]
Theorem 7.3
Theorem 7.1
Theorem 4.3
Kamp [3]
masked
hard-attention
transformers
linear temporal logic

B-RASP: example language

Dyck-1 of depth 2
= strings of parentheses, balanced and nested up to 2 deep
= strings where the number of ℓ's is equal to the number of r's, and every prefix contains 0–2 more ℓ's than r's

Examples:

- accepted: $\ell\ell rr\ell r$ ✓
- accepted: $\ell\ell r\ell rr$ ✓
- rejected: $\ell\ell\ell rrr$ ✗
- rejected: $\ell r\ell r\ell$ ✗



B-RASP: example program

$$
\begin{array}{rcl}
Q_{EOS} & & \\
Q_\ell & & \\
Q_r & & \\
P_\ell(i) & = & \blacktriangleright [j < i, 1] \ \ Q_\ell(j) \\
S_r(i) & = & \blacktriangleleft [j > i, 1] \ \ Q_r(j) \\
I(i) & = & (Q_\ell(i) \wedge S_r(i)) \vee (Q_r(i) \wedge P_\ell(i)) \\
V_\ell(i) & = & \blacktriangleleft [j > i, \neg I(j)] \ \ (Q_\ell(i) \wedge \neg I(i) \wedge \neg Q_r(j)) \\
V_r(i) & = & \blacktriangleright [j < i, \neg I(j)] \ \ (Q_r(i) \wedge \neg I(i) \wedge \neg Q_\ell(j)) \\
Y(i) & = & \blacktriangleleft [1, V_\ell(j) \vee V_r(j)] \ \ \neg(V_\ell(j) \vee V_r(j))
\end{array}
$$

B-RASP: program trace

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}							
Q_ℓ							
Q_r							
P_ℓ							
S_r							
I							
V_ℓ							
V_r							
Y							$\square$

B-RASP: attention operators

$$\blacktriangleright\,[j < i, S(i,j)]\;\; V(j)$$

find rightmost j left of i maximizing $S(i,j)$ and return $V(j)$

B-RASP: attention operators

$$\blacktriangleleft [j > i, S(i,j)] \; V(j)$$

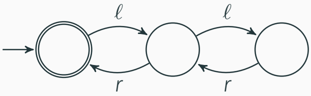
find leftmost j right of i maximizing $S(i,j)$ and return $V(j)$

The initial vectors are Q_{EOS}, Q_ℓ, and Q_r.
These are all defined such that:

$$\boxed{Q_{EOS}(i)} = \text{True iff } \boldsymbol{EOS} \text{ is } i\text{-th symbol}$$

$$\boxed{Q_\ell(i)} = \text{True iff } \ell \text{ is } i\text{-th symbol}$$

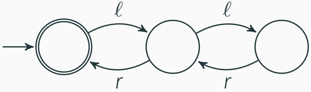
$$\boxed{Q_r(i)} = \text{True iff } r \text{ is } i\text{-th symbol}$$



Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0

P_ℓ - B-RASP Example: Dyck-1 of Depth 2



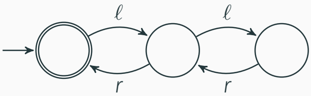
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$\boxed{P_\ell(i)} = \blacktriangleright [j < i, 1] \; Q_\ell(j).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ							

$P_\ell : i = 1$



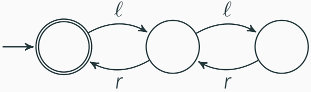
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$\boxed{P_\ell(i)} = \blacktriangleright [j < i, 1] \ \boxed{Q_\ell(j)}\ .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0_j
Q_r	0	0	1	0	1	1	0
P_ℓ	$?_i$						
score	–	–	–	–	–	–	1

$P_\ell : i = 1$



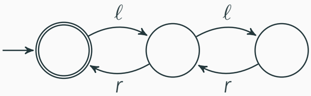
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$\boxed{P_\ell(i)} = \blacktriangleright [j < i, 1] \; \boxed{Q_\ell(j)} \, .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	***EOS***
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0_j
Q_r	0	0	1	0	1	1	0
P_ℓ	0_i						
score	–	–	–	–	–	–	1

$P_\ell : i = 2$



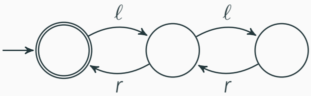
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$\boxed{P_\ell(i)} = \blacktriangleright [j < i, 1]\ \boxed{Q_\ell(j)}\,.$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1_j	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	$?_i$					
score	1_j	–	–	–	–	–	–

$P_\ell : i = 2$



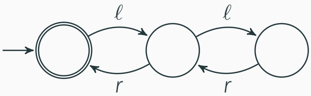
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) \,.$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1_j	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	$?_i$					
score	1_j	–	–	–	–	–	–

$P_\ell : i = 2$



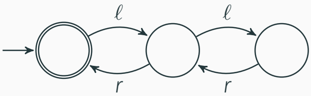
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \ Q_\ell(j) .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1_j	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1_i					
score	1_j	–	–	–	–	–	–

$P_\ell : i = 3$



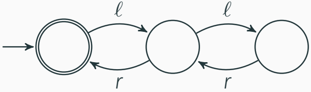
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) \,.$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1_j	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	$?_i$				
score	1	1_j	–	–	–	–	–

$P_\ell : i = 3$



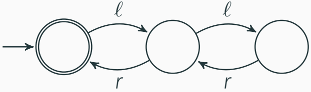
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1_j	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	$?_i$				
score	1	1_j	–	–	–	–	–

$P_\ell : i = 3$



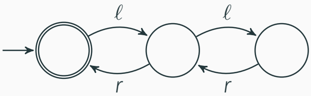
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1_j	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1_i				
score	1	1_j	–	–	–	–	–

$P_\ell : i = 4$



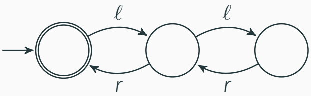
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0_j	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	$?_i$			
score	1	1	1_j	–	–	–	–

$P_\ell : i = 4$



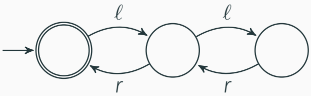
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0_j	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	$?_i$			
score	1	1	1_j	–	–	–	–

$P_\ell : i = 4$



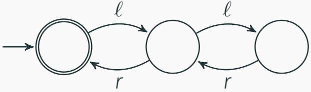
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0_j	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0_i			
score	1	1	1_j	–	–	–	–

$P_\ell : i = 5$



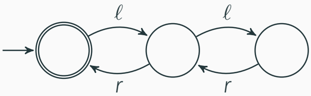
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1_j	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	$?_i$		
score	1	1	1	1_j	–	–	–

$P_\ell : i = 5$



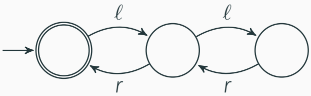
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \ Q_\ell(j) .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1_j	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	$?_i$		
score	1	1	1	1_j	–	–	–

$P_\ell : i = 5$



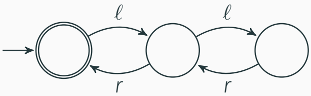
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1_j	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1_i		
score	1	1	1	1_j	–	–	–

$P_\ell : i = 6$



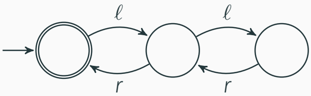
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) \, .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0_j	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	$?_i$	
score	1	1	1	1	1_j	–	–

$P_\ell : i = 6$



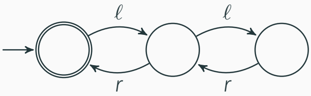
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0_j	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	$?_i$	
score	1	1	1	1	1_j	–	–

$P_\ell : i = 6$



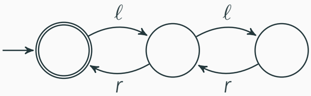
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0_j	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0_i	
score	1	1	1	1	1_j	–	–

$P_\ell : i = 7$



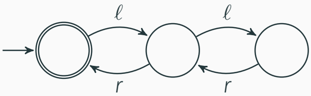
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) \,.$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0_j	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	$?_i$
score	1	1	1	1	1	1_j	–

$P_\ell : i = 7$



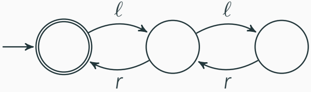
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$P_\ell(i) = \blacktriangleright [j < i, 1] \; Q_\ell(j) \; .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0_j	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	$?_i$
score	1	1	1	1	1	1_j	–

$P_\ell : i = 7$



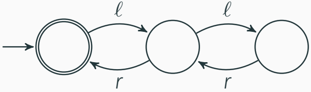
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$\boxed{P_\ell(i)} = \blacktriangleright [j < i, 1] \ \boxed{Q_\ell(j)} .$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0_j	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	0_i
score	1	1	1	1	1	1_j	–

P_ℓ - B-RASP Example: Dyck-1 of Depth 2



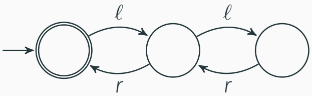
P_ℓ indicates whether the symbol immediately to the left is ℓ.

$$\boxed{P_\ell(i)} = \blacktriangleright [j < i, 1] \; Q_\ell(j).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	0	1	1	0	0

S_r - B-RASP Example: Dyck-1 of Depth 2



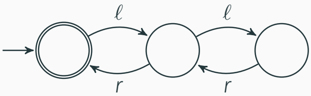
S_r indicates whether the symbol immediately to the right is r.

$$S_r(i) = \blacktriangleleft [j > i, 1] \; Q_r(j).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	0
S_r	0	1	0	1	1	0	0

S_r - B-RASP Example: Dyck-1 of Depth 2



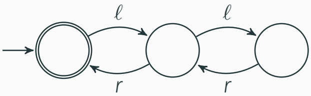
I indicates whether position i is in a consecutive pair ℓr.

$$I(i) = (Q_\ell(i) \wedge S_r(i)) \vee (Q_r(i) \wedge P_\ell(i)).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	0
S_r	0	1	0	1	1	0	0
I	0	1	1	1	1	0	0

V_ℓ - B-RASP Example: Dyck-1 of Depth 2



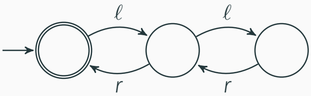
V_ℓ registers a violation for position i if it has symbol ℓ, is not immediately matched, and the next not-immediately-matched symbol is not r:

$$V_\ell(i) = \blacktriangleleft \left[j > i, \neg I(j) \right] \; (Q_\ell(j) \wedge \neg I(j) \wedge \neg Q_r(j)).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	0
S_r	0	1	0	1	1	0	0
I	0	1	1	1	1	0	0
V_ℓ	0	0	0	0	0	0	0

V_r - B-RASP Example: Dyck-1 of Depth 2



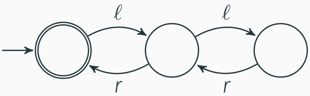
The Boolean vector V_r registers a violation for position i if it has symbol r, is not immediately matched, and the previous not-immediately-matched symbol is not ℓ:

$$V_r(i) = \blacktriangleright \left[j < i, \neg I(j) \right] \; (Q_r(i) \wedge \neg I(i) \wedge \neg Q_\ell(j))$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	*EOS*
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	0
S_r	0	1	0	1	1	0	0
I	0	1	1	1	1	0	0
V_ℓ	0	0	0	0	0	0	0
V_r	0	0	0	0	0	0	0

Y - B-RASP Example: Dyck-1 of Depth 2



The output vector Y checks if there is a violation anywhere.

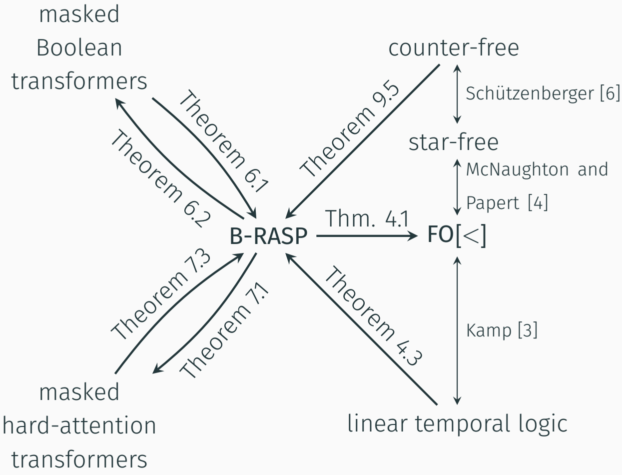
$$Y(i) = \blacktriangleleft \left[1, V_\ell(j) \vee V_r(j)\right] \neg(V_\ell(j) \vee V_r(j))$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	0
S_r	0	1	0	1	1	0	0
I	0	1	1	1	1	0	0
V_ℓ	0	0	0	0	0	0	0
V_r	0	0	0	0	0	0	0
Y	1	1	1	1	1	1	$\boxed{1}$

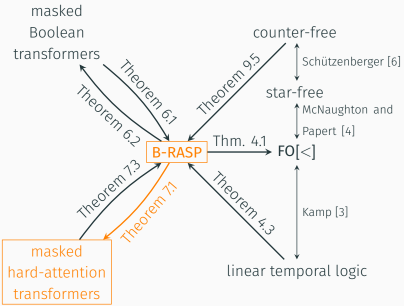
$\ell\ell r\ell rr$: Accepted

Our Results



masked Boolean transformers
counter-free
Schützenberger [6]
star-free
McNaughton and Papert [4]
Theorem 6.1
Theorem 6.2
Theorem 9.5
B-RASP
Thm. 4.1
FO[<]
Theorem 7.3
Theorem 7.1
Theorem 4.3
Kamp [3]
masked hard-attention transformers
linear temporal logic

Results: B-RASP to Transformers



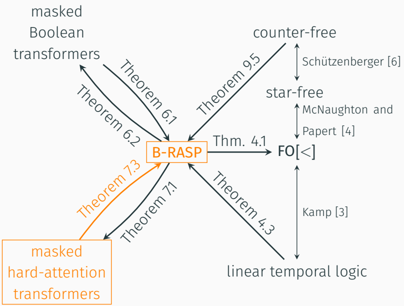
Theorem 7.1
*Any **B-RASP** program can be converted to a masked hard-attention transformer.*

Score predicates $S(i,j)$ can be written in *canonical DNF*:

$$\begin{aligned} S(i,j) = (\alpha_1(i) \wedge \beta_1(j)) \\ \vee (\alpha_2(i) \wedge \beta_2(j)) \\ \vdots \\ \vee (\alpha_k(i) \wedge \beta_k(j)) \end{aligned}$$

which is essentially a dot-product

Results: Transformers to B-RASP

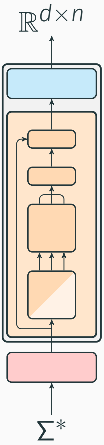


Theorem 7.3
Any masked hard-attention transformer can be converted to a B-RASP program.

Show that the transformer uses only m different activation values. Then represent an activation with $\lceil \log_2 m \rceil$ bits.

Transformer Equivalence

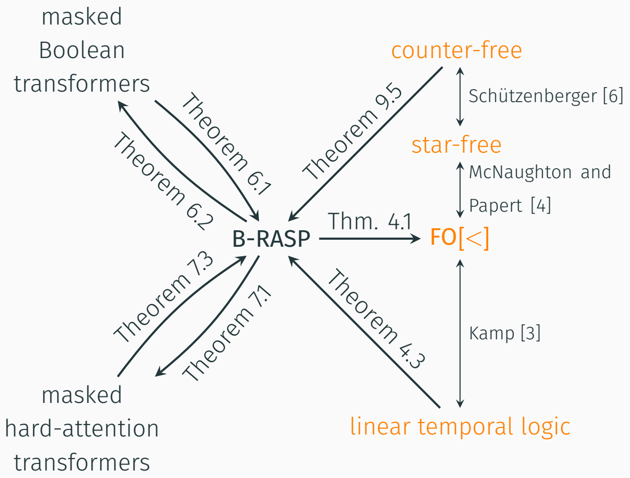
masked hard-attention Transformer



B-RASP

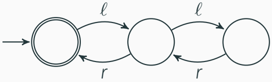
input	ℓ	ℓ	r	ℓ	r	r	EOS
Q_{EOS}	0	0	0	0	0	0	1
Q_ℓ	1	1	0	1	0	0	0
Q_r	0	0	1	0	1	1	0
P_ℓ	0	1	1	0	1	0	0
S_r	0	1	0	1	1	0	0
I	0	1	0	1	0	0	0
V_ℓ	0	0	0	0	0	0	0
V_r	0	0	0	0	0	0	0
Y	1	1	1	1	1	1	1

Our Results


masked
Boolean
transformers
Theorem 6.1
Theorem 6.2
counter-free
Theorem 9.5
Schützenberger [6]
star-free
McNaughton and
Papert [4]
B-RASP
Thm. 4.1
FO[<]
Theorem 7.3
Theorem 7.1
Theorem 4.3
Kamp [3]
masked
hard-attention
transformers
linear temporal logic

Equivalent Formalisms

Counter-free automata



Star-free regular expressions: union, concatenation, complementation

$$\overline{\overline{\emptyset} aa \overline{\emptyset}}$$

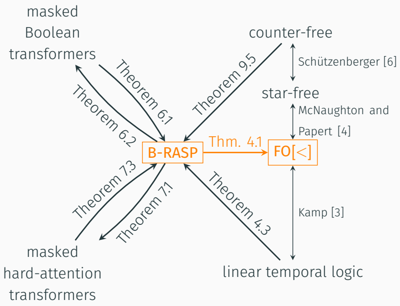
FO[<]: first order logic of strings with order

$$\text{Succ}(i,j) \equiv j > i \wedge \neg(\exists k)(i < k \wedge k < j)$$

LTL: linear temporal logic

$$\phi_1 \wedge \phi_2 \ \textbf{until} \ \neg\phi_3$$

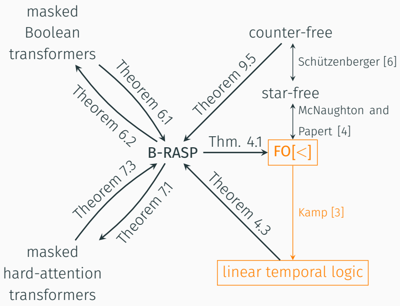
Results: B-RASP to FO[<]



Theorem 4.1
*Any **B-RASP** program can be converted to a **FO**[<] formula.*

This is fairly straightforward!

Prior Work: FO[<] to LTL

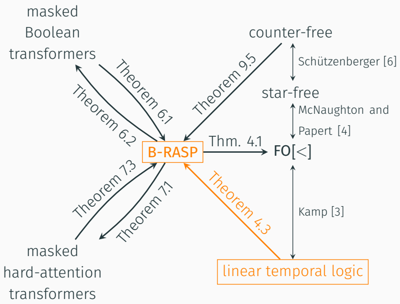


Theorem

Any **FO**[<] *formula can be converted to an LTL formula.*

Originally proved in Kamp's PhD thesis (> 100 pages). Challenge: an **FO**[<] formula has any number of free variables, but an LTL formula has just one

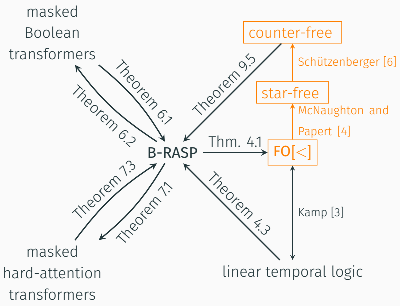
Results: LTL to B-RASP



Theorem 4.3
Any LTL formula can be converted to a ***B-RASP*** *program.*

This is fairly straightforward too: the time variable becomes a position variable.

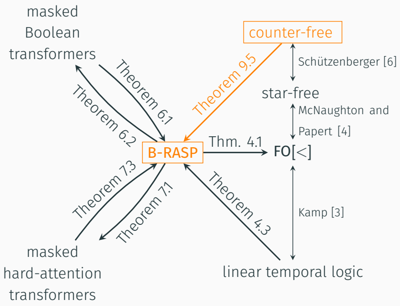
Prior Work: Counter-Free Automata to Star-Free to FO[<]



Theorem *Any formula of* **FO**[<] *can be converted to a star-free regular expression.*

Theorem *Any star-free regular expression can be converted to a counter-free finite automaton.*

Results: Counter-Free Automata to B-RASP



Theorem 9.5 *Any counter-free automaton can be converted to a B-RASP program.*

The Krohn-Rhodes decomposition (one PhD thesis for two people!) is a cascade of *identity–reset* automata, which can be simulated in B-RASP.

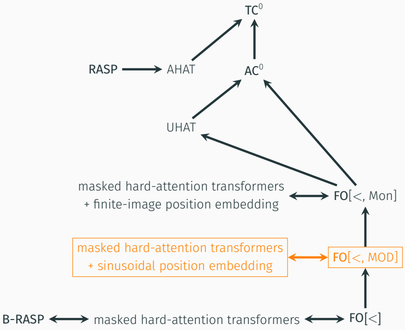
Theorem

B-RASP with strict masking is strictly more expressive than with non-strict masking.

Proof.

No B-RASP program with non-strict masking can recognize the language $\{a\}$ with $\Sigma = \{a\}$. $\square$

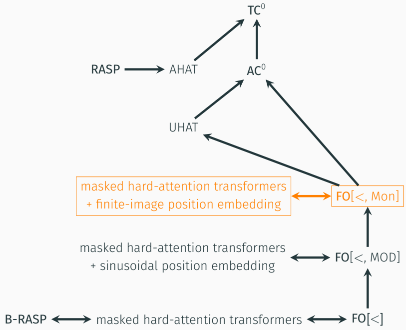
Results: Sinusoidal Positional Embeddings



Corollary *Masked hard-attention transformers with sinusoidal position embeddings recognize exactly the regular languages in* AC^0

By Mix Barrington et al. [5], **FO**$[<, \mathrm{MOD}]$ recognizes exactly the regular languages in AC^0

Results: Positional Embeddings With Finite Image



Corollary *Masked hard-attention transformers that have position embeddings with finite image recognize exactly the languages definable in* **FO**$[<, Mon]$.

What Next?

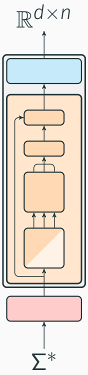
- Average-hard attention?
- Learnability?
- Softmax attention?

Thank You

Stephen Bothwell, Darcey Riley, Ken Sible,

Aarohi Srivastava, Lena Strobl, and Chihiro Taguchi!

Questions?



- Masked hard-attention transformer as a "base case"
- B-RASP and its equivalences
- Strict masking is more powerful than non-strict masking
- Augmenting with position embeddings

References

[1] Pablo Barceló, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. Logical languages accepted by transformer encoders with hard attention, 2023. URL https://arxiv.org/abs/2310.03817. arXiv:2310.03817.

[2] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of Transformers to recognize formal languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, 2020. DOI 10.18653/v1/2020.emnlp-main.576. URL https://aclanthology.org/2020.emnlp-main.576.

[3] Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles, 1968. URL https://www.proquest.com/docview/302320357.

[4] Robert McNaughton and Seymour Papert. *Counter-Free Automata.* Number 65 in M.I.T. Press Research Monographs. The M.I.T. Press, 1971. ISBN 9780262130769. URL https://archive.org/embed/CounterFre_00_McNa.

[5] David A. Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. First-order expressibility of languages with neutral letters or: The Crane Beach conjecture. *Journal of Computer and System Sciences*, 70(2):101–127, 2005. ISSN 0022-0000. DOI https://doi.org/10.1016/j.jcss.2004.07.004. URL https://www.sciencedirect.com/science/article/pii/S0022000004000807.

[6] M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. DOI 10.1016/S0019-9958(65)90108-7.

[7] Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. Transformers as recognizers of formal languages: A survey on expressivity. *arXiv preprint arXiv:2311.00208*, 2023.

[8] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. *arXiv preprint arXiv:2105.11115*, 2021.

Notes on Learnability

Bhattamishra et al. [2] argues that Dyck-1 of depth more than 1 is not learned by transformers

Yao et al. [8] argues that Dyck-k of depth d is learned by transformers for various k and d.