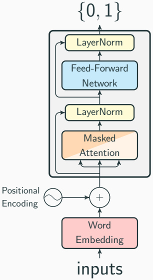
The ABC's of RASP (mostly ~~B's and~~ C's)
due to time

Andy Yang (Univ. of Notre Dame, USA)

11 Oct 2024

Background

Our Perspective: Transformers and Formal Models



$\longleftrightarrow$

$$(\forall i)(Q_{\mathrm{a}})$$

$$(\forall i)(\forall j)(i < j \rightarrow Q_{\mathrm{a}}(i) \wedge Q_{\mathrm{b}}(j))$$

etc.

What languages are recognized by transformer encoders?

What languages are defined by logical formulas?

For a survey of papers in this area: Strobl et al. [2023], "Transformers as Recognizers of Formal Languages: A Survey on Expressivity"

Formal Languages

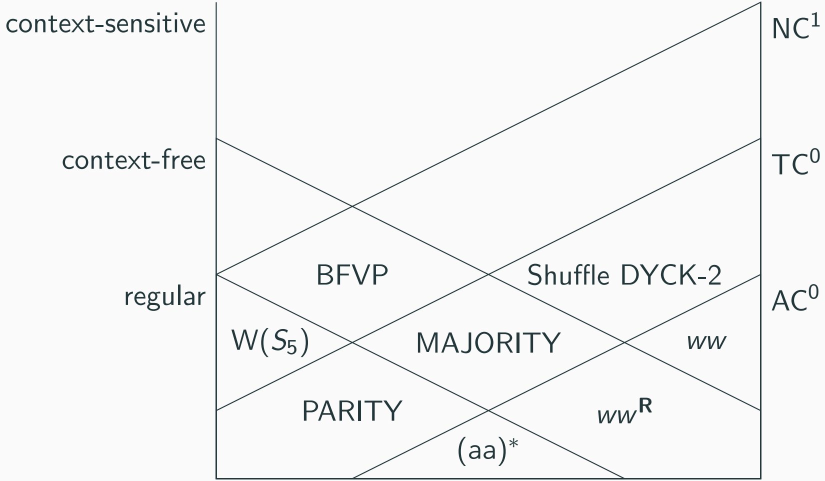


Figure 1: Some complexity classes defined by circuit families, compared with the perhaps more familiar Chomsky hierarchy. See Strobl [2023]

RASP

Symbolic Representation of Transformer Computation

- Matrix multiplications are too confusing for me!
- Can we relate transformers to other formal models?
- How can we prove the expressivity of transformers?

RASP

Challenge 2: Shift

Shift all of the tokens in a sequence to the right by i positions. (Here we introduce an optional parameter in the aggregation: the default value to be used when no input positions are selected. If not defined, this value is 0.)

```
def shift(i=1, default="_", seq=tokens):
    x = (key(indices) == query(indices-i)).value(seq, default)
    return x.name("shift")
shift(2)
```

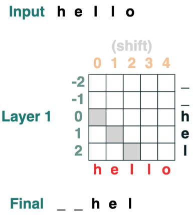


Figure 2: Weiss et al. [2021] and https://srush.github.io/raspy/

Further Work

(subset of) RASP —[Lindner et al., 2024]→ (subset of) transformers

(subset of) RASP ←[Friedman et al., 2024]— (subset of) transformers

Further Work

(subset of) RASP $\xrightarrow{\text{[Lindner et al., 2024]}}$ (subset of) transformers

(subset of) RASP $\xleftarrow{\text{[Friedman et al., 2024]}}$ (subset of) transformers

(subset of) RASP $\xrightarrow{\text{[Yang and Chiang, 2024]}}$ transformers

C-RASP

C-RASP: Counting Operators

$$\# \, [j \leq i] \, P(j)$$

The number of j left of i such that $P(j)$

C-RASP: Example Program for Dyck-1

$$
\begin{array}{rcl}
Q_\ell & & \\
Q_r & & \\
C_\ell(i) & := & \#\,[j \leq i]\;\; Q_\ell(j) \\
C_r(i) & := & \#\,[j \leq i]\;\; Q_r(j) \\
V(i) & := & C_\ell(i) < C_r(i) \\
C_V(i) & := & \#\,[j \leq i]\;\; V(j) \\
M(i) & := & C_V(i) = 0 \\
B(i) & := & C_\ell(i) = C_r \\
D(i) & := & M(i) \wedge B(i)
\end{array}
$$

C-RASP Program Trace

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ						
Q_r						
C_ℓ						
C_r						
V						
C_V						
M						
B						
D						$\square$

Initial Vectors - C-RASP Example: Dyck-1

The initial vectors are Q_ℓ and Q_r. These are all defined such that:

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

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

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T

C_ℓ - C-RASP Example: Dyck-1

C_ℓ counts the number of ℓ seen up until and including current position

$$C_\ell(i) = \#[j \leq i]\ Q_\ell(i).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T
C_ℓ	1	2	2	3	3	3

C_r - C-RASP Example: Dyck−1

C_r counts the number of r seen up until and including current position

$$C_r(i) = \#\,[j \leq i]\;\; Q_r(i).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T
C_ℓ	1	2	2	3	3	3
C_r	0	0	1	1	2	3

V - C-RASP Example: Dyck-1

V indicates a matching violation - if there are ever more r than ℓ

$$V(i) = C_\ell(i) < C_r(i).$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T
C_ℓ	1	2	2	3	3	3
C_r	0	0	1	1	2	3
V	F	F	F	F	F	F

C_V - C-RASP Example: Dyck-1

C_V counts the number of violations seen up until the current position.

$$C_V(i) = \#\,[j \le i]\;\; \text{V(i)}\,.$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T
C_ℓ	1	2	2	3	3	3
C_r	0	0	1	1	2	3
V	F	F	F	F	F	F
C_V	0	0	0	0	0	0

M - C-RASP Example: Dyck-1

M checks that the parentheses are always matched by verifying if there are zero violations

$$M(i) = C_V(i) = 0.$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T
C_ℓ	1	2	2	3	3	3
C_r	0	0	1	1	2	3
V	F	F	F	F	F	F
C_V	0	0	0	0	0	0
M	T	T	T	T	T	T

B - C-RASP Example: Dyck-1

B checks that the parentheses are balanced by verifying if there are equally as many ℓ as r

$$B(i) = C_\ell(i) = C_r(i)$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T
C_ℓ	1	2	2	3	3	3
C_r	0	0	1	1	2	3
V	F	F	F	F	F	F
C_V	0	0	0	0	0	0
M	T	T	T	T	T	T
B	F	F	F	F	F	T

D - C-RASP Example: Dyck-1

D checks the string is matched and balanced.

$$D(i) = M(i) \wedge B(i)$$

Program Trace

input	ℓ	ℓ	r	ℓ	r	r
Q_ℓ	T	T	F	T	F	F
Q_r	F	F	T	F	T	T
C_ℓ	1	2	2	3	3	3
C_r	0	0	1	1	2	3
V	F	F	F	F	F	F
C_V	0	0	0	0	0	0
M	T	T	T	T	T	T
B	F	F	F	F	F	T
D	F	F	F	F	F	T

More Details

Results

- Every **C-RASP** program compiles into a transformer that simulates it perfectly for inputs of arbitrary length
- Allows us to show the expressivity of transformers on many tasks: Dyck-1, $a^n b^n c^n$, and piecewise testable languages.

What is in C-RASP and what isn't?

$\in$ C-RASP

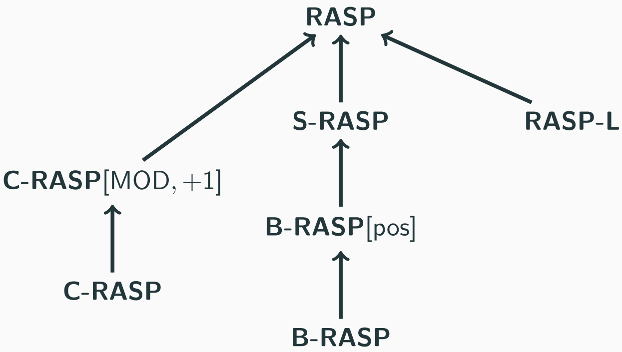
- DYCK-1
- Majority
- $a^n b^n c^n$
- Piecewise testable $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_n \Sigma^*$

$\notin$ C-RASP

- $\Sigma^* a c^* a \Sigma^*$
- $\{a^m \mid m = n^2, n \in \mathbb{N}\}$
- $\{w\$w \mid \text{for } w \in \Sigma\}$
- $(aa)^*$ and PARITY
- $\mathbf{NC}^1$-complete languages

some of these are just conjectures . . .

The RASP Family Tree



Disclaimer: More arrows may exist

That's All

- Every **C-RASP** program compiles into a transformer that simulates it perfectly for inputs of arbitrary length
- Understand what transformer encoders can and cannot do, more intuitively
- Connections with formal language theory and logic

References i

Dan Friedman, Alexander Wettig, and Danqi Chen. Learning transformer programs. *Advances in Neural Information Processing Systems*, 36, 2024.

David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Tom McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *Advances in Neural Information Processing Systems*, 36, 2024.

Lena Strobl. Average-hard attention transformers are constant-depth uniform threshold circuits, 2023. URL `https://arxiv.org/abs/2308.03212`. arXiv:2308.03212.

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.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR, 2021.

Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. In *Proceedings of the Conference on Language Modeling*, 2024. URL `https://arxiv.org/abs/2404.04393`. To appear.

Extra C-RASP Examples

More?

$a^* b^*$ *over* $\Sigma = \{a, b\}$

$C_a(i) := \#[j \leq i] \;\; Q_a(j)$	# positions with *a*'s
$C_b(i) := \#[j \leq i] \;\; Q_b(j)$	# positions with *b*'s
$V(i) := Q_a(i) \wedge C_b(i) \geq 1$	*Violation*: an *a* has *b*'s preceding it
$C_V(i) := \#[j \leq i] \;\; V(j)$	# *Violations*
$Y(i) := C_V(i) = 0$	Zero *Violations*

MORE?

$a^* b^* a^*$ *over* $\Sigma = \{a, b\}$

$C_a(i) := \#[j \leq i]\ \ Q_a(j)$	# positions with *a*'s
$C_b(i) := \#[j \leq i]\ \ Q_b(j)$	# positions with *b*'s
$BA(i) := Q_a(i) \wedge C_b(i) \geq 1$	A subsequence *ba* ends at *i*
$C_{ba}(i) := \#[j \leq i]\ \ BA(j)$	# ends of subsequence *ba*
$BAB(i) := Q_b(i) \wedge C_{ba} \geq 1$	the subsequence *bab* ends at *i*
$C_{bab}(i) := \#[j \leq i]\ \ BAB(j)$	# ends of subsequence *bab*
$Y(i) := C_{bab}(i) = 0$	There are no subsequences *bab*

MORE?!

$a^n b^n c^n$ *over* $\Sigma = \{a, b, c\}$

$C_a(i) := \#[j \leq i]\ \ Q_a(j)$	# positions with a's
$C_b(i) := \#[j \leq i]\ \ Q_b(j)$	# positions with b's
$C_c(i) := \#[j \leq i]\ \ Q_c(j)$	# positions with c's
$A(i) := C_b(i) + C_c(i) = 0$	No preceding b's or c's
$B(i) := C_c(i) = 0$	No preceding c's
$C_A(i) := \#[j \leq i]\ \ Q_a(j) \wedge A(j)$	# a's with no preceding b's or c's
$C_B(i) := \#[j \leq i]\ \ Q_b(j) \wedge B(j)$	# b's with no preceding c's
$G_a(i) := C_A(i) = C_a(i)$	no a's have preceding b's or c's
$G_b(i) := C_B(i) = C_b(i)$	no b's have preceding c's
$G_{abc}(i) := C_a(i) = C_b(i) = C_c(i)$	same number of a's, b's, c's
$Y(i) := G_a(i) \wedge G_b(i) \wedge G_{abc}(i)$	Correct order & number of symbols

Ok one more

hello over $\Sigma = \{e, h, l, o\}$

$C_e(i) := \# [j \leq i] \;\; Q_e(j)$	# positions with *e*'s
$C_h(i) := \# [j \leq i] \;\; Q_h(j)$	# positions with *h*'s
$C_l(i) := \# [j \leq i] \;\; Q_l(j)$	# positions with *l*'s
$C_o(i) := \# [j \leq i] \;\; Q_o(j)$	# positions with *o*'s
$C_\Sigma(i) := \# [j \leq i] \;\; 1$	# symbols in string
$HE(i) := Q_e(i) \wedge C_h(i) = 1$	A subsequence *he* ends at *i*
$C_{he}(i) := \# [j \leq i] \;\; HE(j)$	# ends of subsequence *he*
$HEL(i) := Q_l(i) \wedge C_{he}(i) = 1$	A subsequence *hel* ends at *i*
$C_{hel}(i) := \# [j \leq i] \;\; HEL(j)$	# ends of subsequence *hel*
$HELLO(i) := Q_o(i) \wedge C_{hel}(i) = 2$	A subsequence *hello* ends at *i*
$Y(i) := HELLO(i) \wedge C_\Sigma(i) = 5$	Length 5 and contains *hello*