

Algebraic Decomposition Theory for Transformer Length Generalization

Andy Yang^{1*} Blerta Veseli^{2*} Corentin Barloy³ Michaël Cadilhac⁴

Andreas Krebs⁵ Charles Paperman⁶ Howard Straubing⁷ Michael Hahn²

¹University of Notre Dame, ²Saarland University, ³Ruhr University Bochum

⁴DePaul University, ⁵University of Tübingen, ⁶University of Lille, ⁷Boston College

Abstract

Transformer-based language models are known to sometimes generalize to sequences longer than seen during training, but we lack a precise characterization of which tasks admit length generalization. It is not even known which *regular languages* transformers length-generalize on – and this is a foundational class of languages. Our contributions are to establish the first complete characterization of which regular languages transformers length-generalize on and provide a decision algorithm running in polynomial time in the size of the language’s syntactic monoid. These results rely on an effective characterization of the regular languages in C-RASP, a recently-established formalism that expresses which languages transformers length-generalize on. This characterization is challenging because classical tools like Krohn-Rhodes decomposition theory for finite semigroups are insufficient for C-RASP. Firstly, the basic building blocks of Krohn-Rhodes theory – flip-flop and simple groups – are not expressible in C-RASP. Secondly, the basic building block of C-RASP (unbounded counting) is not expressible by the finite semigroups of Krohn-Rhodes theory. Thus, *length generalization on regular languages is controlled by an algebraic property that is invisible to classical finite decomposition theory*. We generalize classical decomposition theory from finite semigroups to the infinite additive group on the integers, allowing us to characterize C-RASP in terms of iterated wreath products of the integers and derive a provable polynomial-time decision algorithm for regular language membership. Experiments across a broad test suite of regular languages confirm that our theory captures transformers’ length-generalization behavior more accurately than existing classifications.¹

1 Introduction

What state-tracking capabilities do transformer language models possess? To answer this question we study the *regular languages* on which transformers can *length-generalize*. First, regular languages give us a formal framework to describe the structure of state-tracking algorithms that transformers can implement. Second, probing for length-generalization gives us an empirical confirmation that the transformer learns an implementation of the underlying algorithm. Even though this is a fundamental question about an important architecture, we still lack a precise answer. For instance as shown in fig. 1, two structurally similar state-tracking tasks (recognizing $(ab + bbaa)^*$ and $(ab + aabb)^*$) may diverge sharply in terms of length-generalizability – and all existing theory fails to explain this discrepancy.

This line of inquiry has a deep history in machine learning. In fact, Kleene (1956) invented regular expressions specifically for the purpose of analyzing the capabilities of McCulloch-

*AY and BV are co-first authors. Contact: ayang4@nd.edu, {blerta.veseli, mhahn}@uni-saarland.de

¹Code available at [GitHub repository](#).

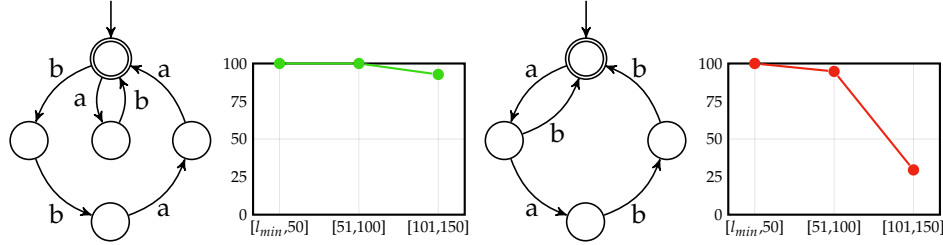


Figure 1: DFAs for $(ab + bbaa)^*$ and $(ab + aabb)^*$ and transformer length generalization on both. Transformers were trained on strings of length $[l_{min}, 50]$, where l_{min} denotes the length of the shortest valid string in the respective language, and the tested on a held-out sets of strings from bins of length up to 150. Existing theory fails to explain why length-generalization differs between these simple regular languages with very similar structure.

Pitts neural networks. Regular language recognition is a fundamental task in the theory of computation that provides us a systematic method of determining the sequence-processing capabilities of a model (Sipser, 1996). The task is simple: given a sequence of inputs, each of which triggers a transition in a deterministic finite automaton (DFA), track the state of the machine after each transition. These finite state-tracking tasks arise concretely in the context of language models, which we sketch below.

Constrained decoding: A user may want a language model to generate text in a structured format, such as JSON or code, which can pose challenges (Schall & de Melo, 2025). Even as outputs grow in length, their structured formats often follow regular constraints (e.g. matching curly braces to a fixed depth in JSON).

Agentic workflows: AI agents in practice follow workflows consisting of compositional sequences of actions (Schluntz & Zhang, 2024). Keeping track of the state of the environment and deciding the next action to take can then explicitly be seen as simulating a DFA.

Natural language: Morphemes in natural languages typically follow constraints that can be modeled by regular languages (Kaplan & Kay, 1994). Natural language semantics also involves maintaining the states of different referents over time (Kim & Schuster, 2023).

Previous work provides a theoretical foundation for our study by providing characterizations of which regular languages can be *expressed* by the transformer architecture. Liu et al. (2023b); Merrill & Sabharwal (2025) showed transformers with $\log(n)$ depth could express all regular languages, while constant depth transformers could express all solvable regular languages. The containment of $\text{poly}(n)$ -precision transformers within TC^0 suggests transformers cannot express non-solvable regular languages (assuming $\text{TC}^0 \neq \text{NC}^1$, as is common) (Merrill & Sabharwal, 2023; Chiang, 2025). Hahn (2020) showed that hard attention transformers recognized only languages in AC^0 , which Yang et al. (2024) later refined to the star-free regular languages, and Jerad et al. (2025) finally refined to the $\mathcal{R}$ -trivial languages (when using leftmost tie breaking). Similarly, Li et al. (2024) showed that finite-precision transformers recognized only star-free languages, which Li & Cotterell (2025) refined to the $\mathcal{R}$ -trivial languages.

In contrast, Bhattamishra et al. (2020); Huang et al. (2025) showed that transformers can *learn* languages both in and outside of each class discussed above, implying that *existing expressivity characterizations do not account for transformer length generalization on regular languages*. In particular, strong empirical evidence show that transformers tend to length-generalize on and only on the languages expressible in C-RASP, a programming language defining a subclass of the languages expressible by a transformer (Huang et al., 2025; Jobanputra et al., 2025; Yang et al., 2025; Yang & Chiang, 2024). However, prior to this work, there did not exist a complete characterization of the regular languages in C-RASP.

Formal language theory has a vibrant tradition of producing beautiful characterizations of regular language membership for different classes. For instance, Barrington et al. (1992)

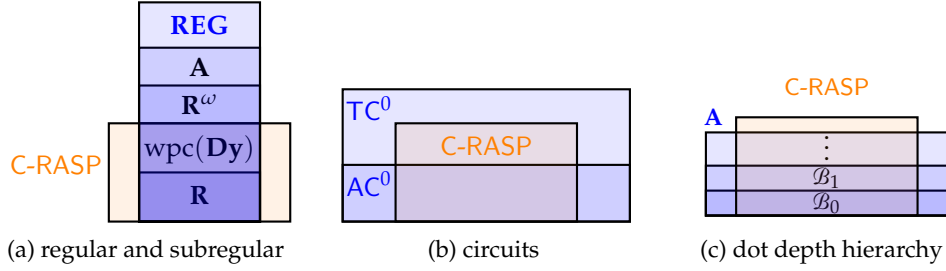


Figure 2: C-RASP is situated orthogonal to well-known classes of languages. C-RASP contains all $\mathcal{R}$ -trivial languages but not all star-free ones (fig. 2a). C-RASP is contained in TC^0 , but incomparable to AC^0 (fig. 2b). C-RASP intersects every level of the dot depth hierarchy, while remaining incomparable to the full hierarchy (fig. 2c).

showed that a regular language is in AC^0 if and only if its syntactic morphism is quasi-aperiodic, thus providing a decision procedure. Simon (1975) showed that a language is piece-wise testable iff its syntactic monoid is $\mathcal{G}$ -trivial. On the frontier, regular language membership in TC^0 is equivalent to the 40 year old open problem of whether or not $TC^0 \neq NC^1$ (Barrington, 1989), and regular language membership in all levels of the dot-depth hierarchy is a 50 year old open problem (Pin, 2017). Our work follows in this tradition, tackling the same question in the case of C-RASP.

This is a deep and challenging theoretical question because classical tools for characterizing regular languages – namely the Krohn-Rhodes decomposition theory of Krohn & Rhodes (1965) – are insufficient for C-RASP. On one front, the building blocks of Krohn-Rhodes theory (flip-flop units and simple groups) are not expressible in C-RASP (Huang et al., 2025). On a second front, the basic building blocks of C-RASP (unbounded counting units) are not expressible by the finite semigroups of Krohn-Rhodes theory. Our core innovation is to develop an analogous algebraic decomposition theory for transformers using the additive group on the integers. We effectively characterize the regular languages in C-RASP, and provide a polynomial-time algorithm that decides DFA membership in C-RASP. A simpler necessary (but not sufficient) criterion is proven via a profinite equation. Finally, we validate empirically that regular language membership in C-RASP predicts transformer length-generalization better than any existing characterization.

2 Algebraic Preliminaries

Our characterization of the regular languages that languages transformers length-generalize on (i.e. those in C-RASP) builds upon the algebraic theory of formal languages (Pin, 2025).

2.1 Basics: Algebraic Theory of Formal Languages

We refer to section C for additional definitions. The essential ones are presented here.

Definition 1 (Monoid). A monoid $(M, \cdot, 1)$ is a set M with an associative binary operation and an identity element. We will just write M when the operation and identity are clear.

A finite monoid M together with a homomorphism $\phi: \Sigma^* \rightarrow M$ behaves like a finite automaton: after reading string $w = w_1 \cdots w_n$, the automaton is in state $\phi(w_1) \cdots \phi(w_n)$. Like a finite automaton, a monoid M recognizes a language L if there exists an accepting subset $X \subseteq M$ and a homomorphism $\phi: \Sigma^* \rightarrow M$ such that $L = \phi^{-1}(X)$. Two monoids will serve as basic units for us (the latter is the “flip-flop” alluded to above).

Definition 2. The monoid U_1 is the set $\{0, 1\}$ where $0 \cdot 1 = 1 \cdot 0 = 0 \cdot 0 = 0$ and $1 \cdot 1 = 1$. The monoid U_2 is the set $\{1, a, b\}$ where $x \cdot a = a$, $x \cdot b = b$, and $1 \cdot x = x = x \cdot 1$ for any $x \in U_2$.

As an example, consider $L = \Sigma^* a \Sigma^*$ and the homomorphism $\phi: \Sigma^* \rightarrow U_1$ where $\phi(\sigma) = 0$ iff $\sigma = a$. Then for any $w \in \Sigma^*$ we have that $\phi(w) = 0 \in U_1$ iff $w \in L$. In some sense, U_1

captures the structure of L (i.e. it detects if a ever occurs in the string). In fact, any monoid recognizing L is divided by U_1 , in the following sense:

Definition 3 (Division). *A monoid M divides a monoid N iff there is a submonoid T of N and homomorphism $\phi: T \rightarrow M$ such that $M = \phi(T)$. Division is transitive.*

For a language L , the *syntactic monoid* $M(L)$ is the unique monoid which recognizes L and divides all other M that also recognize L . In the example above, U_1 is the syntactic monoid of $L = \Sigma^* a \Sigma^*$.

We will consider classes of monoids that enjoy closure properties that will be useful for effective characterizations.

Definition 4 (Pseudovariety). *A pseudovariety of monoids is a collection of monoids closed under division and finite direct products.*

Some pseudovarieties relevant to our characterization are $\mathbf{R}$ (the $\mathcal{R}$ -trivial monoids (Brzozowski & Fich, 1980)), $\mathbf{A}$ (the aperiodic monoids (Schützenberger, 1965)), $\mathbf{REG}$ (all regular languages), and $\mathbf{Dy}$ (the pseudovariety generated by all bounded Dyck monoids).

2.2 Background: Classical Algebraic Decomposition Theory of Regular Languages

The classical algebraic operation for composing monoids is the *wreath product*. Here we will sometimes use $+$ to notate the monoid multiplication to decongest the notation, but we do not intend to suggest it is commutative.

Definition 5 (Classical Wreath Product). *The wreath product $M \circ N$ of finite monoids $(M, +)$ and $(N, \cdot)$ is the monoid $M^N \times N$ with multiplication given by $(f_1, n_1)(f_2, n_2) = (f_1 + {}^{n_1}f_2, n_1 n_2)$, where the left action ${}^n f$ is given by ${}^n f(n') = f(n'n)$.*

The wreath product $M \circ N$ together with a homomorphism $\phi: \Sigma^* \rightarrow M \circ N$ behaves like the composition of a finite automaton and a finite transducer. Let ϕ_M and ϕ_N be such that $\phi(a) = (\phi_M(a), \phi_N(a))$ for all $a \in \Sigma$. After reading a prefix $w_1 \cdots w_t$, the automaton is in state $q_t = \phi_N(w_1) \cdots \phi_N(w_t)$, and the transducer is in state $\phi_M(w_1)(q_0) + \phi_M(w_2)(q_1) + \phi_M(w_3)(q_2) + \cdots + \phi_M(w_t)(q_{t-1})$. Wreath products are the backbone of the fundamental result in the decomposition theory of finite monoids, the Krohn-Rhodes Theorem:

Theorem 6 (Krohn-Rhodes Theorem (Krohn & Rhodes, 1965)). *Every finite monoid M divides an iterated wreath product of U_2 and simple groups G that divide M .*

Unfortunately, Krohn-Rhodes theory is insufficient for handling the monoids involved in transformer length-generalization. C-RASP defines languages with infinite syntactic monoids (Krohn-Rhodes only applies to finite monoids), while U_2 is not definable in C-RASP (Huang et al., 2025) (the Krohn-Rhodes flip-flop unit is useless for C-RASP). The essential monoid for us will be the syntactic monoid of the bounded Dyck monoid, which can be defined in C-RASP.

Definition 7 (Bounded depth Dyck language). *Define $\mathcal{D}_1 := (ab)^*$ and $\mathcal{D}_{k+1} := (a\mathcal{D}_k b)^*$.*

In short: transformers simultaneously succeed at length-generalizing on languages beyond the scope of Krohn-Rhodes theory (Bhattachamishra et al., 2020), and fail to length generalize on the fundamental units in scope of Krohn-Rhodes theory (Liu et al., 2023a). This necessitates a new decomposition theory to handle C-RASP.

3 Algebraic Characterization of C-RASP

Huang et al. (2025) showed that transformer length-generalization can be guaranteed for all languages in C-RASP, and strong empirical evidence suggests failure of length-generalization outside of C-RASP (Jobanputra et al., 2025). Furthermore, a form of fixed-precision transformer is equivalent to C-RASP (Yang et al., 2025). In this section we develop an algebraic characterization of C-RASP, which crucially requires moving beyond Krohn-Rhodes theory to infinite monoids. We refer to Yang & Chiang (2024); Huang et al. (2025) for an exposition of C-RASP and provide a formal definition in section F.

3.1 Typed Monoids

Krebs (2008) developed a framework for using infinite monoids to recognize languages. We present a restriction of the aforementioned framework to the case of wreath products (a one-sided version of the block product used in previous work), which ultimately provides an exact algebraic characterization of C-RASP. The core issue here is that the wreath product of infinite monoids can generate uncountably many elements, which can be too powerful.

Proposition 8. *Consider the classic wreath product $\mathbb{Z} \circ \mathbb{Z}$. Then $M(L) \preceq \mathbb{Z} \circ \mathbb{Z}$ for every L .*

Proof. Without loss of generality let $\Sigma = \{0, 1\}$. Consider the submonoid of $\mathbb{Z} \circ \mathbb{Z}$ generated by the image of Σ^* under the homomorphism $\sigma \mapsto (f_\sigma, 1)$ where $f_\sigma(x) = \sigma \cdot 2^{|x|}$. In essence, this creates a mapping $w \mapsto (f_w, |w|)$ where $f_w(0)$ outputs the integer value of the binary number w . Thus, $\mathbb{Z} \circ \mathbb{Z}$ can recognize arbitrary languages. $\square$

This problem motivates the definition of *typed monoids*, which restricts the accepting sets to be collection of sets closed under union, intersection and complement.

Definition 9. *A typed monoid is a triple $(M, \mathfrak{T}_M, \mathcal{E}_M)$ where M is a finitely generated monoid, $\mathfrak{T}_M$ is a finite Boolean algebra over M , and $\mathcal{E}_M$ is a finite subset of M . Elements of $\mathfrak{T}_M$ are the types and elements of $\mathcal{E}_M$ are the units. A language L is recognized by $(M, \mathfrak{T}_M, \mathcal{E}_M)$ if there exists a homomorphism $h: \Sigma^* \rightarrow M$ such that $h(\Sigma) \subseteq \mathcal{E}_M$ and $L = h^{-1}(\mathfrak{M})$ for some $\mathfrak{M} \in \mathfrak{T}_M$.*

As an example, the language MAJORITY (there are more a 's than b 's) is recognized by the typed monoid $(\mathbb{Z}, \{(-\infty, 0], [1, \infty), \mathbb{Z}, \emptyset\}, \{-1, 1\})$ via the type $[1, \infty)$ and the homomorphism $a \mapsto 1$ and $b \mapsto -1$. We will typically refer to this typed monoid as $\mathbb{Z}$. Now the *typed wreath product* follows the same intuition as the classical wreath product, except the computations are restricted so as not to have more distinguishing power than provided by the types.

Definition 10 (Typed Wreath Product). *Let $(M, \mathfrak{T}_M, \mathcal{E}_M)$, $(N, \mathfrak{T}_N, \mathcal{E}_N)$ be two typed monoids, and let $C \subseteq N$ be a finite set. The typed wreath product*

$$(U, \mathfrak{T}_U, \mathcal{E}_U) = (M, \mathfrak{T}_M, \mathcal{E}_M) \circ_C (N, \mathfrak{T}_N, \mathcal{E}_N)$$

of $(M, \mathfrak{T}_M, \mathcal{E}_M)$ with $(N, \mathfrak{T}_N, \mathcal{E}_N)$ using constants C is defined such that

- $\mathcal{E}_U$ consists of elements (f, n) , where $n \in \mathcal{E}_N$, and $f: N \rightarrow \mathcal{E}_M$ is a type respecting function (see definition 32) with respect to $(N, \mathfrak{T}_N, \mathcal{E}_N)$ and C
- U is the submonoid of $M \circ N$ generated by $\mathcal{E}_U$
- $\mathfrak{T}_U$ consists of types $\mathfrak{U}_{\mathfrak{M}, \mathfrak{N}} = \{(f, n) \mid f(1_N) \in \mathfrak{M}, n \in \mathfrak{N}\}$, where $\mathfrak{M} \in \mathfrak{T}_M, \mathfrak{N} \in \mathfrak{T}_N$

Multiplication is the same as in the classical wreath product.

3.2 Wreath Product Characterization

With the notion of typed monoids in hand, we can give an algebraic characterization of C-RASP. Let $\text{wpc}(M, \mathfrak{T}_M, \mathcal{E}_M)$ denote the *wreath product closure* of a typed monoid, which closes iterated wreath products of this monoid under Boolean combinations and other basic operations. The precise definition will be found in section E.2. The typed wreath product closure turns out to be the precise algebraic “glue” that connects integer counting to C-RASP programs. The proof is given in section F.4.

Theorem 11. $L \in \text{C-RASP} \iff M(L) \in \text{wpc}(\mathbb{Z})$

4 Decomposition Theory for Regular Languages in C-RASP

So far, we have characterized C-RASP in terms of iterated wreath products of $\mathbb{Z}$. We will turn this into an algebraic decision algorithm: given a regular language L , we determine if $M(L)$ divides a wreath product of $\mathbb{Z}$ (or not). A priori, such a question is a formidable problem, due to multiple challenges: $\mathbb{Z}$ is infinite, and we do not even know how many $\mathbb{Z}$ factors are needed. In fact, there even examples in the finite case where the problem

ends up undecidable (Rhodes, 1999). Our second main result will be that, using a detailed understanding of wreath products of $\mathbb{Z}$, this problem is decidable.

In the depth-1 case, it is easy to check if $M(L)$ divides $\mathbb{Z}$ (this is true e.g. for the AND language, $L = 1^*$). How would we check if $M(L) \prec \mathbb{Z} \circ \mathbb{Z}$? If we could “divide” out the right factor to obtain an object “ $M(L)/\mathbb{Z}$ ”, we could check if “ $M(L)/\mathbb{Z}$ ” $\prec \mathbb{Z}$. If this is not the case, we could “divide” by $\mathbb{Z}$ again, and iterate until we either reach division of $\mathbb{Z}$ (i.e., L is in C-RASP), or a fixed point (i.e., L is not in C-RASP). This is the basic idea of our decision procedure. There are three challenges at hand: First, understanding how to “divide” by a wreath product factor; second, making this computable even though $\mathbb{Z}$ is infinite; third, understanding how to identify the fixed point.

4.1 Categories

What do you get when you “divide” one monoid by another? For this first challenge, there is a well-understood technique using categories as algebraic structures (Tilson, 1987). We informally present the main idea, but defer the precise definitions to section G.

In group theory, this question has a simple answer. With a surjective group homomorphism between groups $\phi: G \rightarrow H$, we can “divide” G by $\ker \phi$ with a “quotient” of H , such that $G \preceq (\ker \phi) \circ H$. However, since we are dealing with monoids (and C-RASP does not even contain any non-trivial finite groups), this is of no use for us.

For monoid homomorphisms $\phi: M \rightarrow N$, we may not be able to “divide” M by $\ker \phi$, as monoids’ lack of inverses obstructs such a clean division. What is the divisor in the division $\phi: M(\mathcal{D}_1) \rightarrow U_2$? Intuitively, the wreath product $M(\mathcal{D}_1) \preceq N \circ U_2$ should form a structure that contains elements (last symbol = a) and (last symbol = b). However, composition in this structure will be highly restricted – e.g. (this symbol = b)(last symbol = a) is an illegal composition. It turns out the cleanest way to handle this is by lifting monoids and homomorphisms to higher order structures – categories and relational morphisms.

A category X consists of a set of *objects* $\text{Obj}(X)$ and *hom-sets* $X(x_1, x_2)$, which are collections of *arrows* $\alpha: x_1 \rightarrow x_2$ for each pair of objects $x_1, x_2 \in X$. Arrows can compose associatively, and each object has an identity arrow. A relational morphism of monoids $\phi: M \triangleleft N$ is a relation where $\phi(m) \neq \emptyset$, $\phi(m_1)\phi(m_2) \subseteq \phi(m_1m_2)$, and $1_N \in \phi(1_M)$. This is a generalization of the classical morphism, where elements may map to sets of elements.

Now for any relational morphism of monoids $\phi: M \triangleleft N$, the correct notion of divisor turns out to be the *derived category* D_ϕ . Here, $\text{Obj}(D_\phi) = \phi(M)$ and $D_\phi(n_1, n_2) = \{n_1 \rightarrow_{(m,n)} \mid n \in \phi(m), n_1n = n_2\}$. Arrows compose via the rule $n_0 \rightarrow_{(m_1, n_1)} n_0n_1 \rightarrow_{(m_2, n_2)} = n_0 \rightarrow_{(m_1m_2, n_1n_2)}$, and we merge arrows that have the same behavior under composition.

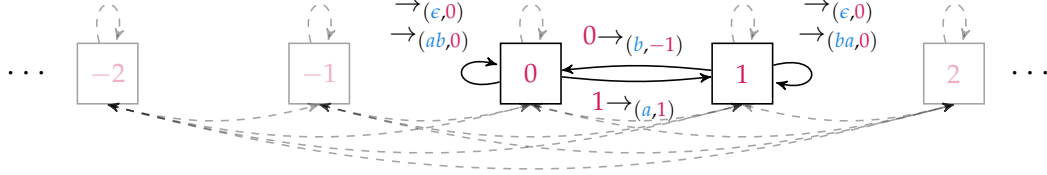
The *Derived Category Theorem* (Tilson, 1987) states that given a relational morphism $\phi: M \triangleleft N$, and a division $D_\phi \preceq V$, we obtain a division $M \preceq V \circ N$. Thus, if $\phi: M \triangleleft \mathbb{Z}$, then D_ϕ formalizes the object “ $M/\mathbb{Z}$ ” alluded to in the previous section. Going forward, we work with an extension of the notions of relational morphisms and division in which the left-hand side can also be a category.

4.2 Decomposition of $\mathcal{D}_1$ into wreath products of $\mathbb{Z}$

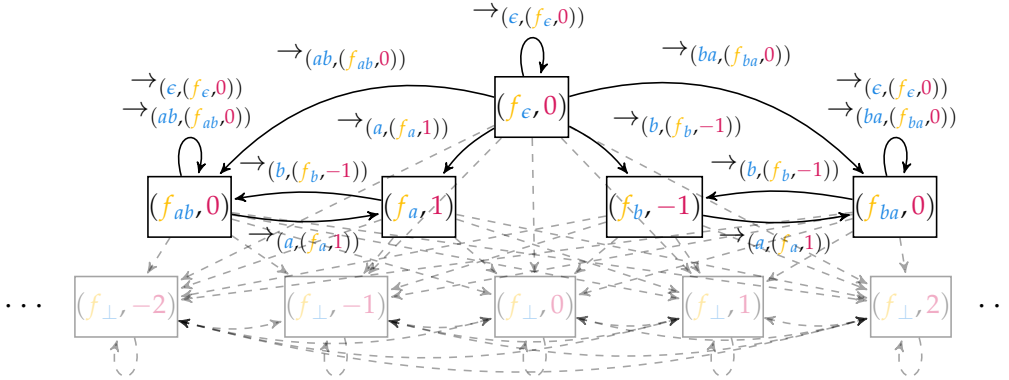
We explain our decision procedure by walking through the division of the syntactic monoid of $\mathcal{D}_1 = (ab)^*$ into an iterated wreath product of $\mathbb{Z}$, via carefully selected relational morphisms into $\mathbb{Z}$. We will visualize objects of a category as rectangles (to prevent confusion with automata) and arrows as labels on edges between squares. Colors indicate the monoid each element comes from. We can view $M(\mathcal{D}_1)$ as a category with a single object and arrows corresponding to monoid elements. We take a relational morphism into $\phi_1: M(\mathcal{D}_1) \triangleleft \mathbb{Z}$ where $\phi_1(\perp) = \mathbb{Z}$, $\phi_1(a) = \{1\}$, $\phi_1(b) = \{-1\}$, $\phi_1(\epsilon) = \phi_1(ab) = \phi_1(ba) = \{0\}$.



In the derived category D_{ϕ_1} there are infinitely many objects, corresponding to each integer. We omit starting or ending objects of arrows where clear by the diagram, combine isomorphic arrows, and let low-opacity arrows and objects denote transitions to $\perp$ elements.



We have not yet arrived at a division $M(\mathcal{D}_1) \preceq \mathbb{Z}$ because D_{ϕ_1} has hom-sets containing multiple arrows (Tilson, 1987, Lemma 3.1, Lemma 4.1). To proceed, we take another relational morphism $\psi_1: D_{\phi_1} \triangleleft \mathbb{Z}$ where $\psi_1(x \rightarrow (\perp, 1)) = \psi_1(x \rightarrow (\perp, -1)) = [1, \infty)$ for all x ; arrows not associated to $\perp$ are instead mapped to $\{0\}$. From ϕ_1, ψ_1 we define a relational morphism $\phi_2: M(\mathcal{D}_1) \triangleleft \mathbb{Z} \circ \mathbb{Z}$ where we let $\phi_2(m) = \left\{ (f_{(m,y)}, y) \mid y \in \phi_1(m), \forall x: f_{(m,y)}(x) = \psi_1(x \rightarrow (m,y)) \right\}$. We visualize the derived category D_{ϕ_2} below (writing (f_m, y) as shorthand for $\phi_2(m)$ whenever it is unique).



We still do not have a division because there are still two arrows in a single homset of D_{ϕ_2} , for instance $\phi_2(\epsilon)$ and $\phi_2(ab)$ act the same on $(f_{ab}, 0)$. To handle this, we define a final relational morphism $\phi_3: M(\mathcal{D}_1) \triangleleft \mathbb{Z} \circ \mathbb{Z} \circ \mathbb{Z}$ such that $\phi_3(m) = (g, \phi_2(m))$ where $g(x) = 0$ iff $x = (f_{(\epsilon, 0)}, 0)$. We ultimately obtain the division. $\phi_3: M(\mathcal{D}_1) \preceq \mathbb{Z} \circ \mathbb{Z} \circ \mathbb{Z}$. In this example, we chose the correct relational morphisms into $\mathbb{Z}$ so as to obtain a division, but in principle there are infinitely many choices of morphisms. Guiding the choice of relational morphisms to reach a termination is a major challenge we address in our decision procedure.

4.3 Algebraic Decision Procedure

We now address the second challenge of maintaining computability despite the existence of infinitely many relational morphisms into $\mathbb{Z}$. First, we divide $M(L)$ into equivalence classes via the $\mathcal{R}$ relation, a classical relation in semigroup theory which characterizes which elements are reachable by other elements via right-multiplication (Pin, 2025). We construct a sequence of iterated relational morphisms into $\mathbb{Z}$ by iterating over the $\mathcal{R}$ -classes in order. At step $i + 1$, we assume that a relational morphism ϕ_i covers all $\mathcal{R}$ -classes up to R_i , and extend to a relational morphism ϕ_{i+1} that covers R_{i+1} . This is done by iteratively computing nontrivial relational morphisms whose values are bounded on R_i . This process terminates, because the set of such bounded relational morphisms forms a finitely generated $\mathbb{Z}$ -module (an integer analogue of vector spaces), and every step finds a morphism linearly independent of the previous ones.

Finally we address the termination conditions by proving completeness of the algorithm. If the algorithm terminates with success, we have constructed a division from $M(L)$ into a wreath product of $\mathbb{Z}$ (and thus C-RASP by theorem 11). Otherwise, if the algorithm terminates with failure, we show no such division exists. In this case, assume for sake of contradiction that $M(L)$ divides a T -fold iterated wreath product of $\mathbb{Z}$ via a relational morphism $\psi: M(L) \preceq \mathbb{Z} \circ \dots \circ \mathbb{Z}$ (where T is minimal). The rightmost component defines a relational morphism $\omega: M(L) \triangleleft \mathbb{Z}$. Either ω is bounded, and it would have been chosen as part of the decomposition had it provided any useful information, or ω is unbounded which renders long strings indistinguishable by types in $\mathbb{Z}$ since values may run to infinity. In either case, we can remove the rightmost component of ψ , and obtain a division from M into a $(T - 1)$ -fold iterated wreath product of $\mathbb{Z}$, contradicting the minimality of T .

To formalize the above proof, we choose relational morphisms to $\mathbb{Z}$ which truncate values beyond a threshold k (which results in relational morphisms to $\mathcal{D}_k$). This allows us to formalize the proof using finite category theory (avoiding the use of types on the category side). A formal writeup can be found in section H.3.

4.4 Necessary (but not Sufficient) Condition via Equations

In this section use profinite equations to give an alternative characterization of the regular languages in C-RASP. These equations are a technique drawing ideas from topology to derive elegant and decidable characterizations for classes of monoids. We defer a precise definition to Pin (2009).

Definition 12. Define $\mathbf{R}^\omega$ as an equation and the corresponding variety of monoids² as

$$\mathbf{R}^\omega : (xy^\omega)^\omega x = (xy^\omega)^\omega.$$

For our purposes, a finite monoid M satisfies $\mathbf{R}^\omega$ if for any $m_1, m_2 \in M$ we have that $(m_1 m_2^\omega)^\omega = (m_1 m_2^\omega)^\omega m_1$, where m^ω denotes m^k such that $m^k = m^{2k}$ (i.e. the unique idempotent generated by m). We give an exact characterization of the monoids in $\mathbf{R}^\omega$, the proof of which is in section I.

Theorem 13. $M \in \mathbf{R}^\omega$ iff M is aperiodic and every $\mathcal{R}$ -class of M contains at most one idempotent.

Equivalently, $\mathbf{R}^\omega = \mathbf{R} \circ \mathbf{G} \cap \mathbf{A}$, where $\mathbf{R}$ denotes the monoids where each $\mathcal{R}$ -class is singleton, $\mathbf{G}$ denotes the finite groups, and $\mathbf{A}$ denotes the aperiodic monoids. In our experiments below, we will draw languages from the larger class $\mathbf{R} \circ \mathbf{G}$ rather than $\mathbf{R}^\omega$ for testing length-generalization. While both the main decision procedure and $\mathbf{R}^\omega$ require iterating over the $\mathcal{R}$ -classes of the monoid, the former needs to iteratively compute relational morphisms, while the latter only needs to count idempotents. This gives a much simpler criterion that is *necessary* for membership in C-RASP, though we can show it is *not sufficient*.

4.5 Algebraic Characterization

The decision procedure results in an exact algebraic characterization of the regular languages in C-RASP – namely, they are wreath products of bounded-depth Dyck languages. We thus derive a small hierarchy within the classes of finite monoids surrounding C-RASP.

Theorem 14. $\text{C-RASP} \cap \mathbf{REG} = \text{wpc}(\mathbf{Dy})$. Hence $\mathbf{R} \subsetneq \text{C-RASP} \cap \mathbf{REG} \subsetneq \mathbf{R}^\omega \subsetneq \mathbf{A} \subsetneq \mathbf{REG}$

A proof is given in section I. Furthermore, the decision procedure runs in polynomial time in the size of the monoid. This allows us to efficiently decide whether or not we expect a transformer to length-generalize on any given regular language. A proof is in section H.3.

Theorem 15. Membership of $M \in \text{C-RASP} \cap \mathbf{REG}$ is decidable in $O(\text{poly}(|M|))$ time.

5 Experiments

We empirically evaluate whether our characterization of regular languages in C-RASP predicts transformer length-generalization. In particular, we examine the different levels

² $\mathbf{R}^\omega$ alludes to the equation for $\mathcal{R}$ -trivial monoids, $(xy)^\omega x = (xy)^\omega x$ (Almeida & Azevedo, 1989)

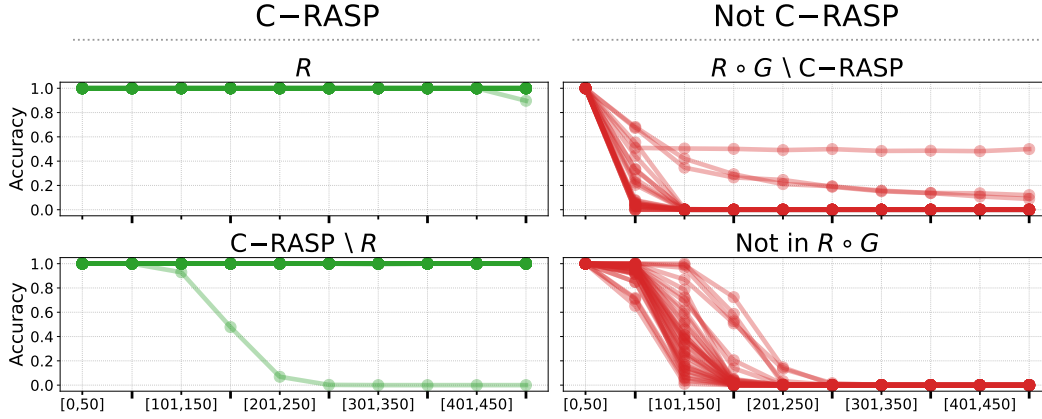


Figure 3: **Length generalization on regular languages.** Models are trained on strings of lengths in $[l_{\min}, 50]$, and evaluated on bins $[l_{\min}, 50]$ (in-distribution) up to $[451, 500]$ in bins of width 50. Each curve corresponds to a language. Green curves denote languages in C-RASP, while red curves denote languages not in C-RASP. Languages in C-RASP maintain near-perfect accuracy well beyond the training range, whereas languages outside C-RASP exhibit rapid degradation, typically failing shortly after. Each panel includes all languages in the corresponding class from Table 1 in appendix B.3.

of the hierarchy shown in theorem 14 to determine the efficacy of each class in predicting length-generalization by transformers. Because $\mathbf{R}^\omega \setminus \mathbf{C-RASP} \cap \mathbf{REG}$ contains few samples, we report results based on membership in $\mathbf{R} \circ \mathbf{G}$ (whose aperiodic fragment is $\mathbf{R}^\omega$).

5.1 Languages

We construct a diverse suite of 125 regular languages, including both representative examples from prior work (Li & Cotterell, 2025; Huang et al., 2025) and systematically generated ones. We provide the full table of languages in Table 1, appendix B.3). To generate languages within the classes discussed in theorem 14, we sample regular expressions using a probabilistic context-free grammar (PCFG) for which we tune the probabilities to obtain diverse class membership. For each language, we determined membership in C-RASP using an automata-based version of the algebraic procedure, which we explain in section J.

5.2 Experimental Setup

Task Definition. The task is state prediction, i.e. tracking automaton states over prefixes. Let $w = a_1 \dots a_n$, and denote by $w_{1:i} = a_1 \dots a_i$ the prefix of length i . Each symbol a_i triggers a transition, and the model must predict the sequence of DFA states $q_1, \dots, q_n$, where q_i is the state reached after processing $w_{1:i}$.

Training and Evaluation. Per formal language, we train GPT-2 models on 10,000 words sampled from the training length range $[l_{\min}, 50]$ with $l_{\min}$ being the length of the shortest valid word in the respective language. We use an 80/20 train-test split. We then evaluate generalization on test length ranges $[51, 100], [101, 150] \dots [451, 500]$, with 1,000 words in each test set. We define successful length generalization as maintaining near in-distribution performance at lengths beyond $2 \times$ the maximum training length 50. Following Huang et al. (2025) we used AdamW with weight decay 0.01 and dropout 0.0. We performed a hyperparameter sweep over layers $\{1, 2, 4\}$, heads $\{1, 2, 4\}$, dimension $\{16, 64, 256\}$ and learning rates $\{0.001, 0.0001\}$ – training every combination in the grid with early stopping when 100% accuracy is achieved on in-distribution test data. For those languages, where in-distribution accuracy never reached 100%, we additionally performed a hyperparameter sweep over layers $\{6, 8, 12\}$, heads $\{4, 8\}$, dimensions $\{64, 256\}$ and learning rates $\{0.001, 0.0001\}$. We adopted optimization choices and hyperparameter ranges from Huang et al. (2025). Per

language, we choose the configuration that achieves the highest accuracy on the longest test length range among those whose accuracy on in-distribution length $[l_{min}, 50]$ is 100%. We break remaining ties by successively considering the next-longest ranges and finally preferring the smallest model (fewest layers, then attention heads, then hidden dimension). The chosen configuration was then used for a multi-seed run, in which models were trained under different random model initializations, an approach commonly used in prior work (Li & Cotterell, 2025; Huang et al., 2025). A run was considered successful if it achieved 100% accuracy on the in-distribution test set. We evaluated random initializations sequentially, up to a maximum of 1,000 trials, and ended the search once five successful runs were found. We report the best successful seed in Figure 3 and show the average performance across all successful seeds in B.

Architectural Constraints. To align empirical results with our theoretical setup, state prediction should depend on the evolving prefix rather than local information. We therefore introduce two constraints: first, we remove positional information by replacing positional embeddings with the zero function (NoPE) forcing the model to rely on the sequential order of tokens provided by the causal attention mask. Second, while NoPE removes absolute positional information, the model still has direct access to the most recent input symbol a_i when predicting the target state q_i , which provides a positional signal through alignment with the prediction target. This can enable shortcuts in which the model predicts the state based solely on the final symbol, rather than the full prefix. To prevent this, we insert a separator token & between symbols (see below) and require the model to predict the state q_i only at these separator positions. Consequently, access to the most recent symbol is also mediated through the attention mechanism. Word symbols receive placeholder targets # and are excluded from the cross-entropy loss. Each target sequence begins with the initial DFA state, predicted at the first separator following <bos>.

Input	<BOS> & a & b & a & b & a & b & <EOS>														
Positional Encoding	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Target	#	1	#	3	#	1	#	3	#	1	#	3	#	1	#

5.3 Results

Figure 3 shows that C-RASP membership is a strong predictor of transformer length-generalization. Languages in C-RASP generalize reliably to substantially longer lengths, while languages outside C-RASP fail to do so, with accuracy rapidly collapsing beyond the training length. This demonstrates that across all evaluated languages C-RASP provides an accurate characterization of length generalization.

6 Conclusion

We have precisely characterized the regular languages on which transformers length-generalize using a decision procedure for regular language membership in C-RASP (which runs in polynomial-time). Experiments on a range of languages in and outside of C-RASP demonstrate that the theory accurately predicts when transformers do and do not length-generalize. The decision procedure is based on a novel algebraic decomposition theory for C-RASP that is distinguished from the classical Krohn-Rhodes theory due to the inclusion of infinite monoids and the omission of the aperiodic flip-flop unit U_2 . A simpler criterion which is necessary (but not sufficient) for membership in C-RASP is also given in the form of a profinite equation. Our results provide a deeper understanding of the state-tracking capabilities of transformers on the machine learning front, as well as a deeper understanding of counting, on the algebraic front.

Acknowledgments

We thank Dana Angluin, Michael Benedikt, David Chiang, and Will Merrill for fruitful discussion and feedback. We thank the anonymous reviewers for their helpful comments.

Funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK 2853/1 “Neuroexplicit Models of Language, Vision, and Action” - project number 471607914, and the US National Science Foundation (grant number 2502292). AY is supported by the US National Science Foundation Graduate Research Fellowship Program under Grant No. 2236418. MH acknowledges support from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project number 560456343.

References

- Jorge Almeida and Assis Azevedo. The join of the pseudovarieties of R-trivial and L-trivial monoids. *Journal of Pure and Applied Algebra*, 60(2):129–137, 1989. ISSN 0022-4049. doi: [https://doi.org/10.1016/0022-4049\(89\)90125-4](https://doi.org/10.1016/0022-4049(89)90125-4). URL <https://www.sciencedirect.com/science/article/pii/0022404989901254>.
- Eric Alsmann, Lowejatan Noori, and Martin Lange. On the expressiveness of state space models via temporal logics. In *The Fourteenth International Conference on Learning Representations*, 2026. URL <https://openreview.net/forum?id=Vg511oJScS>.
- David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989. ISSN 0022-0000. doi: [https://doi.org/10.1016/0022-0000\(89\)90037-8](https://doi.org/10.1016/0022-0000(89)90037-8). URL [https://doi.org/10.1016/0022-0000\(89\)90037-8](https://doi.org/10.1016/0022-0000(89)90037-8).
- David A. Mix Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. Regular languages in NC¹. *Journal of Computer and System Sciences*, 44(3):478–499, 1992. ISSN 0022-0000. doi: [https://doi.org/10.1016/0022-0000\(92\)90014-A](https://doi.org/10.1016/0022-0000(92)90014-A). URL [https://doi.org/10.1016/0022-0000\(92\)90014-A](https://doi.org/10.1016/0022-0000(92)90014-A).
- Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid. Typed monoids – an eilenberg-like theorem for non regular languages. In Franz Winkler (ed.), *Algebraic Informatics*, pp. 97–114, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-21493-6. doi: https://doi.org/10.1007/978-3-642-21493-6_6.
- Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7096–7116, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.576. URL <https://aclanthology.org/2020.emnlp-main.576/>.
- J.A. Brzozowski and Faith E. Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980. ISSN 0022-0000. doi: [https://doi.org/10.1016/0022-0000\(80\)90003-3](https://doi.org/10.1016/0022-0000(80)90003-3).
- David Chiang. Transformers in uniform TC⁰. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL <https://openreview.net/forum?id=ZA7D4nQuQF>.
- Mai Gehrke and Andreas Krebs. Stone duality for languages and complexity. *ACM SIGLOG News*, 4(2):29–53, May 2017. doi: 10.1145/3090064.3090068. URL <https://doi.org/10.1145/3090064.3090068>.
- Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 01 2020. ISSN 2307-387X. doi: 10.1162/tacl_a_00306. URL https://doi.org/10.1162/tacl_a_00306.

- Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A formal framework for understanding length generalization in transformers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=U49N5V51rU>.
- Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 977–996, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-252-7. doi: 10.18653/v1/2025.acl-short.76. URL <https://aclanthology.org/2025.acl-short.76/>.
- Mayank Jobanputra, Yana Veitsman, Yash Sarrof, Aleksandra Bakalova, Vera Demberg, Ellie Pavlick, and Michael Hahn. Born a transformer – always a transformer? On the effect of pretraining on architectural abilities. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=Huw15LqglI>.
- Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8(4):499–507, 1979. doi: 10.1137/0208040. URL <https://doi.org/10.1137/0208040>.
- Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994. URL <https://aclanthology.org/J94-3001/>.
- Najoung Kim and Sebastian Schuster. Entity tracking in language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3835–3855, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.213. URL <https://aclanthology.org/2023.acl-long.213/>.
- S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pp. 3–42. Princeton University Press, Princeton, 1956. ISBN 9781400882618. doi: doi:10.1515/9781400882618-002. URL <https://doi.org/10.1515/9781400882618-002>.
- Andreas Krebs. *Typed Semigroups, Majority Logic, and Threshold Circuits*. PhD thesis, Universität Tübingen, 2008. URL <https://nbn-resolving.org/urn:nbn:de:bsz:21-opus-36244>. URN: urn:nbn:de:bsz:21-opus-36244.
- Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965. doi: 10.1090/S0002-9947-1965-0188316-1. URL <https://doi.org/10.1090/S0002-9947-1965-0188316-1>.
- Jiaoda Li and Ryan Cotterell. Characterizing the expressivity of fixed-precision transformer language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=29LwAgLFpj>.
- Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=3EWTEy9MTM>.
- Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Exposing attention glitches with flip-flop language modeling. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL <https://openreview.net/forum?id=VzmpXQAn6E>.
- Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023b. URL <https://openreview.net/forum?id=De4FYqjFueZ>.

- William Merrill and Ashish Sabharwal. A logic for expressing log-precision transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 52453–52463. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a48e5877c7bf86a513950ab23b360498-Paper-Conference.pdf.
- William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=5pHfYe10iX>.
- Jean-Eric Pin. Profinite Methods in Automata Theory. In Susanne Albers and Jean-Yves Marion (eds.), *26th International Symposium on Theoretical Aspects of Computer Science*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 31–50, Dagstuhl, Germany, 2009. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-939897-09-5. doi: 10.4230/LIPIcs.STACS.2009.1856. URL <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.STACS.2009.1856>.
- Jean-Eric Pin. The dot-depth hierarchy, 45 years later. In Stavros Konstantinidis, Nelma Moreira, Rogério Reis, and Jeffrey Shallit (eds.), *The Role of Theory in Computer Science - Essays Dedicated to Janusz Brzozowski, The Role of Theory in Computer Science - Essays Dedicated to Janusz Brzozowski*. World Scientific, 2017. doi: 10.1142/9789813148208_0008. URL <https://hal.science/hal-01614357>.
- Jean-Éric Pin. Mathematical foundations of automata theory. Lecture notes, MPRI, IRIF, CNRS and Université de Paris. Available at <https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf>, 2025.
- John Rhodes. Undecidability, automata, and pseudovarieties of finite semigroups. *International Journal of Algebra and Computation*, 9(3):455–474, 1999. doi: 10.1142/S0218196799000278. URL <https://doi.org/10.1142/S0218196799000278>.
- Yash Sarrof, Yana Veitsman, and Michael Hahn. The expressive capacity of state space models: A formal language perspective. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=eV5YIrJPdy>.
- Maximilian Schall and Gerard de Melo. The hidden cost of structure: How constrained decoding affects language model performance. In Galia Angelova, Maria Kunilovskaya, Marie Escribe, and Ruslan Mitkov (eds.), *Proceedings of the 15th International Conference on Recent Advances in Natural Language Processing - Natural Language Processing in the Generative AI Era*, pp. 1074–1084, Varna, Bulgaria, September 2025. INCOMA Ltd., Shoumen, Bulgaria. URL <https://aclanthology.org/2025.ranlp-1.124/>.
- Erik Schluntz and Barry Zhang. Building effective agents. <https://www.anthropic.com/engineering/building-effective-agents>, December 2024. Accessed: 2026-03-23.
- M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965. doi: [https://doi.org/10.1016/S0019-9958\(65\)90108-7](https://doi.org/10.1016/S0019-9958(65)90108-7).
- Imre Simon. Piecewise testable events. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, pp. 214–222, Berlin, Heidelberg, 1975. Springer-Verlag. ISBN 3540074074. doi: https://doi.org/10.1007/3-540-07407-4_23.
- Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996. ISBN 053494728X. URL <https://dl.acm.org/doi/10.5555/524279>.
- Price Stiffler Jr. Extension of the fundamental theorem of finite semigroups. *Advances in Mathematics*, 11(2):159–209, 1973. doi: [https://doi.org/10.1016/0001-8708\(73\)90007-8](https://doi.org/10.1016/0001-8708(73)90007-8).
- Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi: <https://doi.org/10.1137/0201010>.
- Denis Thérien and Thomas Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. *SIAM Journal on Computing*, 31(3):777–798, 2001. doi: <https://doi.org/10.1137/S0097539797322772>.

- Bret Tilson. Categories as algebra: An essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48(1):83–198, 1987. ISSN 0022-4049. doi: [https://doi.org/10.1016/0022-4049\(87\)90108-3](https://doi.org/10.1016/0022-4049(87)90108-3).
- Sam van der Poel, Dakotah Lambert, Kalina Kostyszyn, Tiantian Gao, Rahul Verma, Derek Andersen, Joanne Chau, Emily Peterson, Cody St. Clair, Paul Fodor, Chihiro Shibata, and Jeffrey Heinz. Mlregtest: A benchmark for the machine learning of regular languages. *Journal of Machine Learning Research*, 25(283):1–45, 2024. URL <http://jmlr.org/papers/v25/23-0518.html>.
- Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=FmhPg4UJ9K>.
- Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=FBMsBdH0yz>.
- Andy Yang, Michaël Cadilhac, and David Chiang. Knee-deep in c-RASP: A transformer depth hierarchy. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=jPduiyxyfw>.

Contents

1	Introduction	1
2	Algebraic Preliminaries	3
2.1	Basics: Algebraic Theory of Formal Languages	3
2.2	Background: Classical Algebraic Decomposition Theory of Regular Languages	4
3	Algebraic Characterization of C-RASP	4
3.1	Typed Monoids	5
3.2	Wreath Product Characterization	5
4	Decomposition Theory for Regular Languages in C-RASP	5
4.1	Categories	6
4.2	Decomposition of $\mathcal{D}_1$ into wreath products of $\mathbb{Z}$	6
4.3	Algebraic Decision Procedure	7
4.4	Necessary (but not Sufficient) Condition via Equations	8
4.5	Algebraic Characterization	8
5	Experiments	8
5.1	Languages	9
5.2	Experimental Setup	9
5.3	Results	10
6	Conclusion	10
A	FAQ	16
B	Additional Experimental Results	19
B.1	Additional Results on the Main Language Suite	19
B.1.1	Results by C-RASP Membership	19
B.1.2	Length Generalization Across Seeds and Languages	19
B.1.3	Increased Training Data Size	20
B.2	Experiments on More Complex Languages	21
B.2.1	Length Generalization	21
B.2.2	Results by C-RASP Membership	22
B.2.3	Length Generalization Across Seeds and Languages	22
B.3	Regular Languages	23
C	Algebraic Preliminaries	23
D	Linear RASP programs	26

E Typed Monoids	27
E.1 Definitions	28
E.2 Proof of Typed Wreath Product Principle	30
F C-RASP	32
F.1 Definitions	32
F.2 C-RASP as a pseudovariety of Languages	33
F.3 Existing Characterizations	34
F.4 Algebraic Characterization of C-RASP	34
G Derived Categories	34
H Algebraic Decision Procedure	36
H.1 Relevant Lemmas	36
H.2 Bounded-Depth Dyck Monoids	36
H.3 Decidability Proof (via Finite Derived Categories)	37
H.3.1 Key Properties of the Procedure	39
H.3.2 Establishing Completeness	42
H.3.3 Deriving Main Results	46
H.4 Defining Division between Categories and Typed Monoids	47
I Algebraic Characterization (Necessary but not Sufficient Criterion)	48
J Automata proof of $C\text{-RASP} \cap \text{REG}$	49
K Implementation	53

Author Contributions

AY led paper writing, drafted the proof of Theorem 11 and the implementation of the decision procedure, contributed to the discovery and proof of Theorems 14 and 15, drafted Appendix J, and drafted Sections 1–4 of the paper. BV designed, implemented, and carried out the experiments, and drafted Section 5 of the paper. CB, MC, AK, CP, HS contributed to the discovery and proof of Theorems 14 and 15, and provided input to the paper writing. HS further contributed Theorem 13. MH contributed to the discovery and proof of Theorems 14 and 15, drafted the proof of Theorems 14 and 15 in Appendix H and an early draft of Appendix J, and contributed to paper writing.

A FAQ

1. *Q: How does the work relate and compare to Liu et al. (2023b)?*
Liu et al. (2023b) primarily looked at expressivity, not length generalization. Indeed, their experiments also confirmed that transformers may not length-generalize on the particular regular languages that they were able to express and learn to a fixed length.
Our results may also apply to the expressive power of transformers under a particular fixed-precision assumption, because C-RASP was shown to be equivalent to these transformers by Yang et al. (2025).

2. Q: How about other architectures? Like log-depth transformers, state-space models, ...?

It is already known that log-depth architectures can simulate arbitrary automata, so a characterization of the regular languages they can express is not needed (Liu et al., 2023b; Merrill & Sabharwal, 2025). As for length-generalization, we do not presently know what regular languages these architectures length-generalize on.

Analyzing architectures with limited recurrence, like state-space models, would be an interesting future question. Under varying assumptions, state-space models are able to simulate flip-flops and counting (Sarraf et al., 2024; Alsmann et al., 2026). The techniques we have developed to handle counting could be extended to handle these cases as well.

3. Q: Why not evaluate LLMs?

We’re more interested in the architecture itself. LLM abilities depend on prompt format and are strongly impacted by what’s in the training data. There is work suggesting that the capabilities of LLMs are ultimately bounded by C-RASP (Jobanputra et al., 2025), though a precise investigation in the case of regular language length-generalization is out of the scope of this work.

4. Q: How does C-RASP compare to other classes, such as subregular classes, circuit classes, and the dot depth hierarchy? Could they on their own already predict transformer length generalization?

C-RASP is distinct from known classes, and much more successful at predicting length generalization than those are. Here, we will expand on fig. 2. In fact, all existing classes do not predict length-generalization on transformers.

- Regular and subregular: The regular languages in C-RASP define a strict subset of the star-free languages. As we show, even $\mathbf{R}^\omega$ only covers star-free languages (and in fact it is a strict subset).
- Circuit classes: The circuit classes most relevant to transformers are AC^0 and TC^0 ; neither of them captures C-RASP or transformer length generalization well. C-RASP is a strict subset of TC^0 , as was shown in Huang et al. (2025). This extends to regular languages: E.g., the PARITY language ($b^*ab^*ab^*$) (checking if the number of a ’s is even) is in TC^0 but not in C-RASP. In general, C-RASP is incomparable with AC^0 . Interestingly, on the level of regular languages, C-RASP is a strict subset of AC^0 : C-RASP only includes star-free languages (and all of those are in AC^0), but, for instance, $\{a, b\}^*b$ is in AC^0 but not in C-RASP.
- Dot-depth hierarchy. Given the containment inside the star-free languages, one could consider stratifications of these languages. One of the most well-studied ones is the dot-depth hierarchy. C-RASP can express every bounded-depth dyck languages, and thus can touch every single level of the dot-depth hierarchy, while not covering the entire hierarchy.

5. Q: What about the role of positional encodings?

This is an important question for future work that would result in a different characterization than the one we have derived here. Huang et al. (2025) showed that C-RASP[periodic, local] characterizes the languages which transformers with APE can length-generalize on. Obtaining a decision procedure on the algebraic side would require additional techniques from the ones used in this paper.

6. Q: Why did you not use MLRegTest for your experiments (van der Poel et al., 2024)?

We looked into this, but unfortunately a significant portion of the languages in the dataset contain *strictly local* patterns, which are not expressible in C-RASP without positional encodings (Huang et al., 2025). Future work extending the characterization of C-RASP to handle positional encodings should use this benchmark.

7. Q: The experiments deliberately prevent residual-stream access to the last token in state prediction. Why? Could the theory handle the case where one actually wants this access?

We argue that state tracking should be robust to the insertion of “do-nothing” actions not changing the state, or other unrelated extra material. In formal language theory, such actions are referred to as *neutral symbols*. Invariance under the inclusion of such neutral symbols is a natural property of characterizations in terms of syntactic

monoids, as we have done here. An example where this plays a role is $\{a, b\}^*b$ (where the state – *is the last symbol seen a b?* – can be easily tracked based on the last symbol), which has the same syntactic monoid as the language $\{a, b, e\}^*be^*$ resulting from adding a neutral symbol e , which is difficult for Transformer length generalization (Liu et al., 2023a; Huang et al., 2025). Neither is definable in C-RASP.

8. Q: Why do your empirical results differ from that of Li & Cotterell (2025) and Huang et al. (2025) who both test length generalization for regular languages in C-RASP?

We think addressing the empirical and theoretical results of Huang et al. (2025) and Li & Cotterell (2025) in light of our findings is an important question. Li & Cotterell (2025) suggested that transformers consistently fail to length-generalize on languages outside of $\mathbf{R}$, providing evidence when trained on length N and tested on length $12N$. While Huang et al. (2025) showed length-generalization in languages in $\text{C-RASP} \setminus \mathbf{R}$, they only trained on length N and tested up to $3N$, so the results may be incomparable. In the present work, we specifically probed languages in $\text{C-RASP} \cap \mathbf{REG} \setminus \mathbf{R}$ and found length-generalization from length N to $10N$, suggesting a more optimistic picture of length-generalization than Li & Cotterell (2025).

We suspect the difference in our empirical observations might be attributed to the small number of languages in $\text{C-RASP} \cap \mathbf{REG} \setminus \mathbf{R}$ tested by Li & Cotterell (2025) – only 3 such languages were tested. Indeed, prior to this work there existed no sizeable dataset of languages in $\text{C-RASP} \cap \mathbf{REG} \setminus \mathbf{R}$, as there was no computable membership criterion for $\text{C-RASP} \cap \mathbf{REG}$.

An additional difference is that Li & Cotterell (2025) use a language *classification* task where the model is trained on both positive and negative examples. The experiments done by Bhattamishra et al. (2020); Huang et al. (2025); Yang et al. (2025), and this work on C-RASP all use a *next-token prediction* setup.

B Additional Experimental Results

In this section, we provide additional experimental results supporting Section 5. We first report complementary analyses on the language suite from Table 1 used for the experiments in Figure 3 shown in the main paper. Specifically, we present best seed results grouped directly by C-RASP membership in Section B.1.1, results across multiple random seeds in Section B.1.2, and experiments with increased training data (from 10K to 100K) in Section B.1.3.

We then evaluate a second suite of more complex regular languages with greater nesting depth (Table 2). For these languages, we extend the training range from $[l_{min}, 50]$ to $[l_{min}, 200]$ and report the corresponding experiments in Section B.2. We present length generalization results across the classes $\mathbf{R}$, C-RASP, and $\mathbf{R} \circ \mathbf{G}$ in Section B.2.1, directly compare languages in and outside of C-RASP in Section B.2.2, and evaluate performance across multiple random seeds in Section B.2.3.

Finally, Section B.3 provides the complete lists of regular languages used in both experimental suites, together with their membership in $\mathbf{R}$, $\mathbf{R}^\omega$, $\mathbf{R} \circ \mathbf{G}$, and C-RASP.

B.1 Additional Results on the Main Language Suite

B.1.1 Results by C-RASP Membership

The main paper reports length generalization results separately for languages in $\mathbf{R}$, C-RASP $\setminus \mathbf{R}$, $\mathbf{R} \circ \mathbf{G} \setminus \text{C-RASP}$, and outside $\mathbf{R} \circ \mathbf{G}$ (Figure 3). Here, we provide an alternative view of the same experimental results by grouping languages according to their C-RASP membership only. Figure 4 shows a clear separation between languages, where languages in C-RASP reliably generalize beyond the training length range, whereas languages outside C-RASP consistently fail to do so.

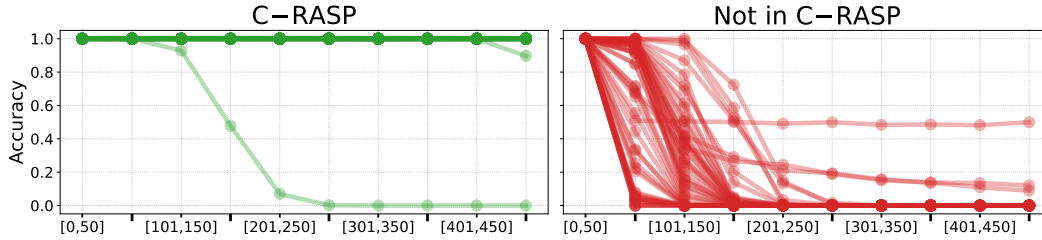


Figure 4: **Length generalization by C-RASP membership.** Models are trained on strings of lengths in $[l_{min}, 50]$ and evaluated on ranges from $[l_{min}, 50]$ (in-distribution) up to $[451, 500]$ in steps of 50. Each curve corresponds to a language from Table 1.

B.1.2 Length Generalization Across Seeds and Languages

The results in the main paper report the best successful seed for each language. To assess whether the observed length generalization behavior is robust across random seeds (i.e. random model initializations), we additionally aggregate results over the five best successful seeds per language, where a seed is considered successful if it achieves 100% accuracy on the in-distribution test data. Figure 5 reports these results across $\mathbf{R}$, C-RASP $\setminus \mathbf{R}$, $\mathbf{R} \circ \mathbf{G} \setminus \text{C-RASP}$, and outside $\mathbf{R} \circ \mathbf{G}$, while Figure 6 groups them only by C-RASP membership. Results show that languages outside C-RASP consistently fail to length generalize across languages and across random seeds.

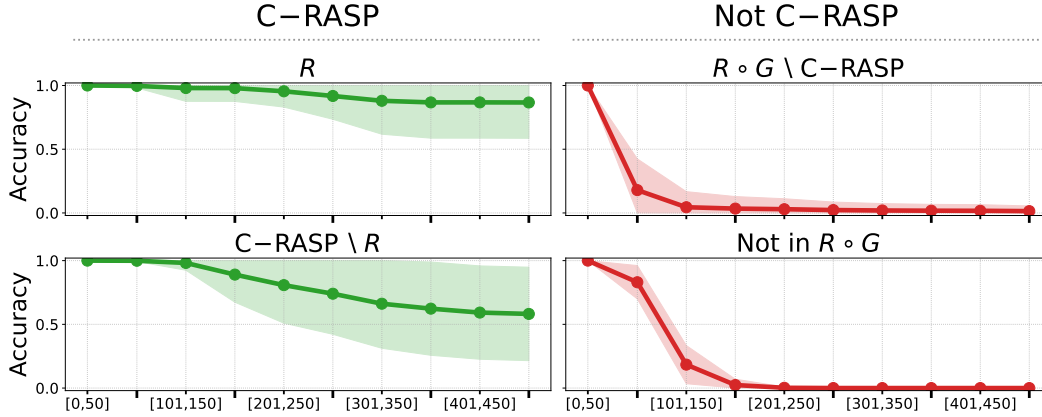


Figure 5: **Length generalization on regular languages (aggregated across languages and seeds).** Models are trained on strings of lengths in $[l_{min}, 50]$ and evaluated on length ranges from $[l_{min}, 50]$ (in-distribution) up to $[451, 500]$ in steps of 50. For each language we compute the mean accuracy across its 5 best successful seeds, where a seed is considered successful if it achieves 100% accuracy on the in-distribution test data. Solid lines show the mean of these per-language curves within each group, and shaded regions show one standard deviation across languages. Green corresponds to languages in C-RASP, while red corresponds to languages not in C-RASP. Each panel includes all languages in the corresponding group from Table 1.

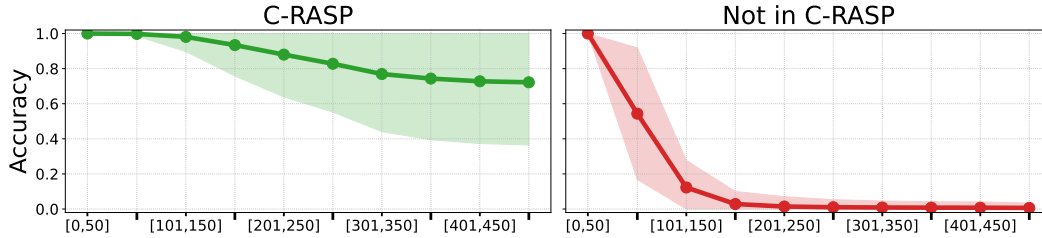


Figure 6: **Length generalization by C-RASP membership, aggregated across languages and seeds.** Models are trained on strings of lengths in $[l_{min}, 50]$ and evaluated on length ranges from $[l_{min}, 50]$ (in-distribution) up to $[451, 500]$ in steps of 50. For each language, we compute the mean accuracy across its 5 best successful seeds, where a seed is considered successful if it achieves 100% accuracy on the in-distribution test data. Solid lines show the mean of these per-language curves for languages within and outside C-RASP, and shaded regions show one standard deviation across languages. In contrast to Figure 5, which separates languages into R , $C-RASP \setminus R$, $R \circ G \setminus C-RASP$, and those outside $R \circ G$, here we group the same languages by C-RASP membership only, providing an overall view of length generalization within and outside C-RASP. Each panel includes all languages in the corresponding group from Table 1.

B.1.3 Increased Training Data Size

Finally, we test how increased training data size affects length generalization within different classes. We repeat the experiments with 100K sampled words per language, using an 80/20 train-test split, compared to 10K training examples in the original experiments. We otherwise follow the same training and hyperparameter search procedure described in Section 5. Figure 7 shows that increasing the amount of training data does not change trends in length generalization: languages in C-RASP continue to generalize beyond the training lengths, while languages outside C-RASP do not. In Figure 8 we again regroup the same results from Figure 7 into languages in and outside of C-RASP only.

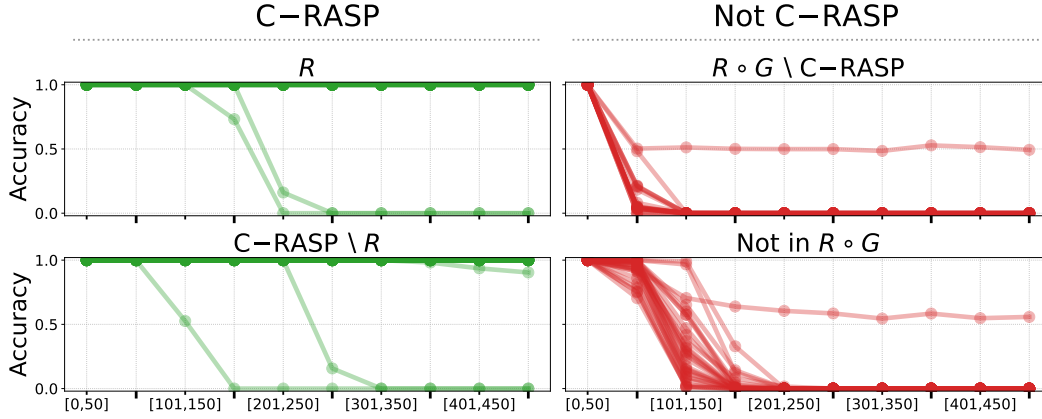


Figure 7: **Length generalization on regular languages with increased training data.** Models are trained on strings of lengths in $[l_{min}, 50]$ using a larger training set (100K examples instead of 10K), and evaluated on lengths from $[l_{min}, 50]$ (in-distribution) up to $[401, 500]$ in steps of 50. Each curve corresponds to a language. We report the best seed per language. Increasing the training data does not change trends in length generalization: languages in C-RASP continue to generalize, while languages not in C-RASP consistently fail to generalize beyond the training range. For these experiments, we used the systematically generated subset of languages listed in Table 1, consisting of 100 languages in total.

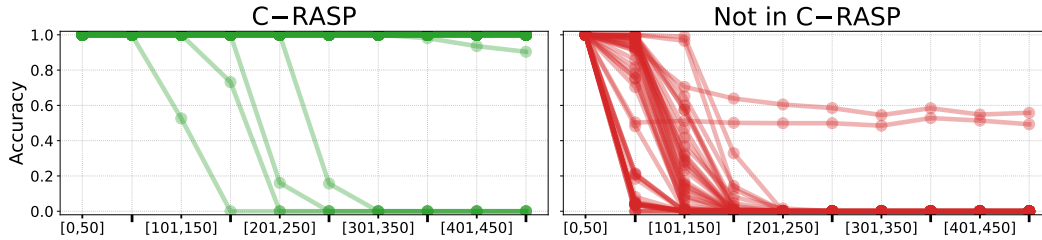


Figure 8: **Length generalization by C-RASP membership with increased training data.** Models are trained on strings of lengths in $[l_{min}, 50]$ using a larger training set (100K examples instead of 10K) and evaluated on length ranges from $[l_{min}, 50]$ (in-distribution) up to $[401, 500]$ in steps of width 50. Each curve corresponds to a language, and we report the best seed per language. In contrast to Figure 7, which separates languages into the four groups $\mathbf{R}$, $\mathbf{C-RASP} \setminus \mathbf{R}$, $\mathbf{R} \circ \mathbf{G} \setminus \mathbf{C-RASP}$, and languages outside $\mathbf{R} \circ \mathbf{G}$, here we group the same languages solely by C-RASP membership, providing an overall view of length generalization within and outside C-RASP. For these experiments, we used the systematically generated subset of languages listed in Table 1, consisting of 100 languages in total.

B.2 Experiments on More Complex Languages

To test whether our findings extend to more complex languages, we repeat our experiments on a systematically generated set of 50 regular languages with greater nesting depth (Table 2), following the same experimental procedure described in Section 5. In addition, we increase the maximum training length from 50 to 200, allowing us to examine length generalization behavior when models are trained on longer strings.

B.2.1 Length Generalization

In Figure 9 we show results across $\mathbf{R}$, $\mathbf{C-RASP} \setminus \mathbf{R}$, $\mathbf{R} \circ \mathbf{G} \setminus \mathbf{C-RASP}$, and those outside $\mathbf{R} \circ \mathbf{G}$, confirming length generalization trends seen on simpler languages and shorter training lengths. Figure 10 highlights this length generalization trends further by dividing languages into subsets of in and outside of C-RASP providing a more summarized overview.

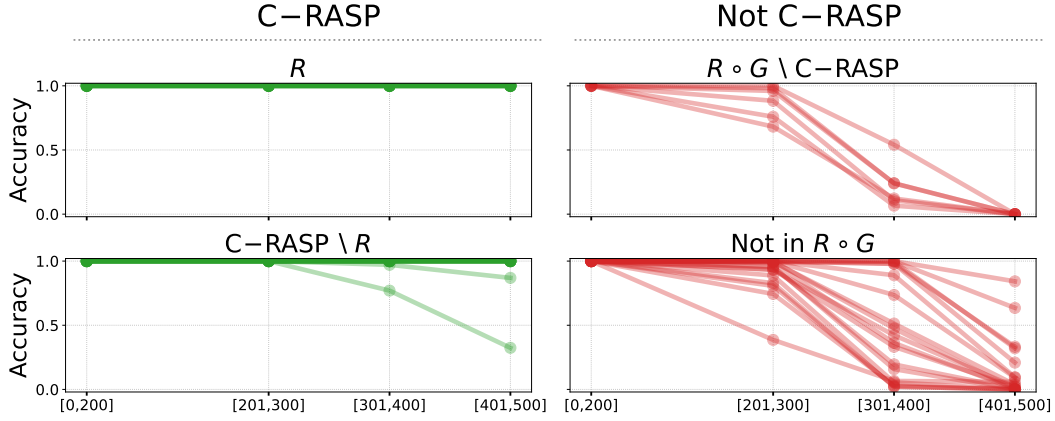


Figure 9: **Length generalization on regular languages (longer training length)**. Models are trained on strings of lengths in $[l_{min}, 200]$, and evaluated on length ranges $[l_{min}, 200]$ (in-distribution) up to $[401, 500]$ in steps of 100. Each curve corresponds to a language. Green curves denote languages in C-RASP, while red curves denote languages not in C-RASP. Languages in C-RASP maintain near-perfect accuracy well beyond the training range, whereas languages outside C-RASP exhibit rapid degradation, typically failing shortly after. Languages corresponding to this plot can be viewed in Table 2.

B.2.2 Results by C-RASP Membership

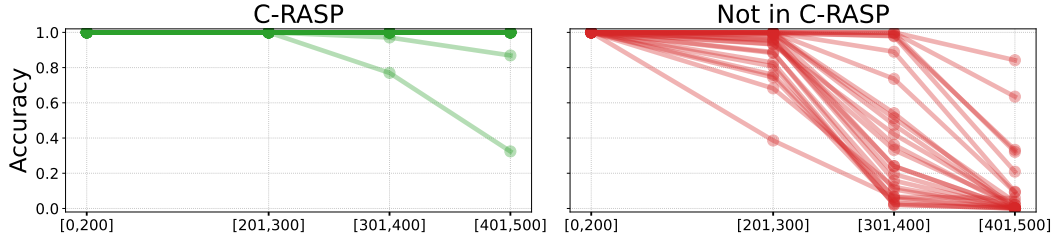


Figure 10: **Length generalization by C-RASP membership (longer training length)**. Models are trained on strings of lengths in $[l_{min}, 200]$ and evaluated on length ranges from $[l_{min}, 200]$ (in-distribution) up to $[401, 500]$ in steps of 100. Each curve corresponds to a language. In contrast to Figure 9, which separates languages into the four groups R , $C-RASP \setminus R$, $R \circ G \setminus C-RASP$, and languages outside $R \circ G$, here we group the same languages by C-RASP membership, providing an overall view of length generalization within and outside C-RASP. Each panel includes all languages in the corresponding group from Table 2.

B.2.3 Length Generalization Across Seeds and Languages

Similar to Section B.1.2, we evaluate how consistent length generalization trends are across random seeds (i.e. random model initializations) and languages within a class. We compute the mean accuracy across the five best successful seeds for each language and aggregate these results across languages. Figure 11 shows that same length generalization trends persist across seeds: languages in C-RASP maintain high accuracy beyond the training length range, while languages outside C-RASP consistently degrade with increasing length. Figure 12 further emphasizes this difference by grouping languages in and outside of C-RASP.

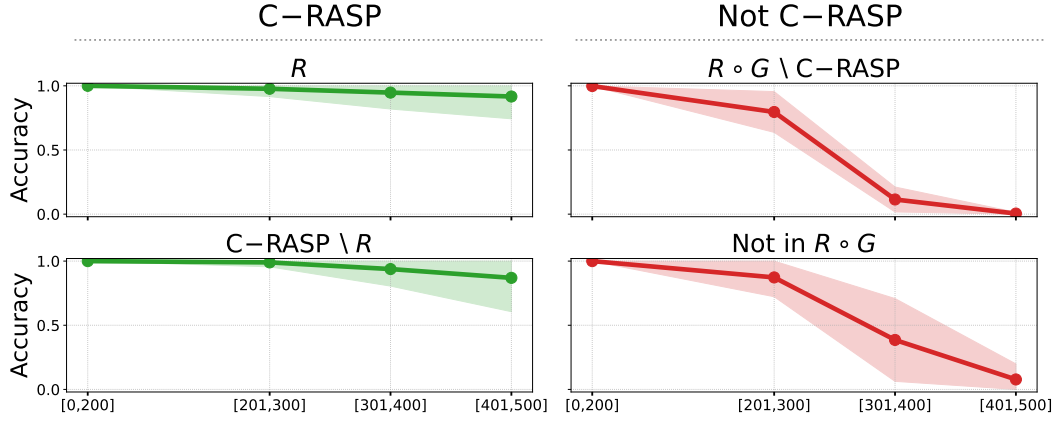


Figure 11: **Length generalization on regular languages (aggregated across languages and seeds).** Models are trained on strings of lengths in $[l_{min}, 200]$ and evaluated on length ranges from $[l_{min}, 200]$ (in-distribution) up to $[401, 500]$ in steps of 100. For each language we compute the mean accuracy across its 5 best successful seeds, where a seed is considered successful if it achieves 100% accuracy on the in-distribution test data. Solid lines show the mean of these per-language curves within each class, and shaded regions show one standard deviation across languages. Green corresponds to languages in C-RASP, while red corresponds to languages not in C-RASP. Results for individual best seeds are shown in Figure 9. Each panel includes all languages in the corresponding class from Table 2.

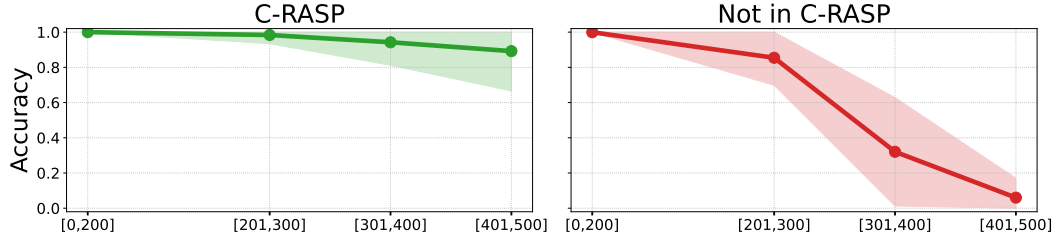


Figure 12: **Length generalization by C-RASP membership, aggregated across languages and seeds.** Models are trained on strings of lengths in $[l_{min}, 200]$ and evaluated on length ranges from $[l_{min}, 200]$ (in-distribution) up to $[401, 500]$ in steps of 100. For each language, we compute the mean accuracy across its 5 best successful seeds, where a seed is considered successful if it achieves 100% accuracy on the in-distribution test data. Solid lines show the mean of these per-language curves for languages within and outside C-RASP, and shaded regions show one standard deviation across languages. In contrast to Figure 11, which separates languages into the four groups R , $C-RASP \setminus R$, $R \circ G \setminus C-RASP$, and languages outside $R \circ G$, here we group the same languages solely by C-RASP membership, providing an overall view of length generalization within and outside C-RASP. Each panel includes all languages in the corresponding group from Table 2.

B.3 Regular Languages

We report the complete set of regular languages used in our experimental evaluation in the main paper and appendix. Tables 1 and 2 provide a comprehensive overview of the dataset. For each language, we indicate membership in R , R^ω , $R \circ G$, and C-RASP.

C Algebraic Preliminaries

Definition 16 (Recognition; Syntactic Monoid). *Let $L \subseteq \Sigma^*$ be a language. A monoid M recognizes L iff there is a homomorphism $h: \Sigma^* \rightarrow M$ and a subset $X \subseteq M$ such that $L = h^{-1}(X)$.*

Formal Language	R	C-RASP	R^ω	$R \circ G$	Formal Language	R	C-RASP	R^ω	$R \circ G$
$(bbac)^*$	False	True	True	True	$(ab)^* + (bb)^*$	False	False	False	True
$(bab + b)^*$	False	False	False	False	$(bb)^*(bb)^*$	False	False	False	True
$(c + b(a)^*)^*$	False	False	False	False	$((b)^*ac)^*$	False	False	False	False
$b + (bb)^*$	False	False	False	True	$(bc(c)^*)^*$	False	False	False	False
$acabcc(c)^*$	True	True	True	True	$(baa + a)^*$	False	False	False	False
$bbcc(aa)^*$	False	False	False	True	$(ac)^*c + ba + a$	False	True	True	True
$(ab(a)^*)^*$	False	False	False	False	$(b + a)^*aaac$	False	False	False	False
$(cc)^*$	False	False	False	True	$(bb)^*cbac$	False	False	False	True
$((a)^*ac)^*$	False	False	False	False	$((b)^*)^*(ab)^*$	False	True	True	True
$((b)^*ab)^*$	False	False	False	False	$bba(c + a)^*$	True	True	True	True
$(ac(a)^*)^*$	False	False	False	False	$cb(a)^*baa$	True	True	True	True
$(cca + a)^*$	False	False	False	False	$(b)^*baa$	True	True	True	True
$cb(a)^* + abbc$	True	True	True	True	$(aa + (b)^*)^*$	False	False	False	True
$aa(a)^*c$	True	True	True	True	$aa + ca(aa)^*$	False	False	False	True
$(aa + aa)^*$	False	False	False	True	$(a)^*(a)^*b + aac$	True	True	True	True
$(a + c)^*cbbb$	False	False	False	False	$(bb)^*ab + ac$	False	False	False	True
$bc(b)^*abbb$	True	True	True	True	$(cc)^*cccb$	False	False	False	True
$(ca)^*(cb)^*$	False	True	True	True	$(b)^*cbccaa$	True	True	True	True
$(cc)^*ccb + b$	False	False	False	True	$((b)^*)^*b^*$	True	True	True	True
$((a)^*)^*(ac)^*$	False	True	True	True	$ccc(ba)^*$	False	True	True	True
$bccb(aa)^*$	False	False	False	True	$bcac(a)^* + ba$	True	True	True	True
$(ba)^*(b)^*bc$	False	True	True	True	$(b)^*cc(ca)^*$	False	True	True	True
$(ac)^*ba(b)^*$	False	True	True	True	$(cab + c)^*$	False	False	False	False
$(a)^*b + acab$	True	True	True	True	$(a + ac + a)^*$	False	False	False	False
$(ca)^*(a)^*bb$	False	True	True	True	$aba + c(bb)^*$	False	False	False	True
$(b)^*(b)^*(aa)^*$	False	False	False	True	$((a)^*ca)^*$	False	False	False	False
$b + ca + b(c + a)^*$	True	True	True	True	$(abbc)^*$	False	True	True	True
$(cbb + b)^*$	False	False	False	False	$(a + c)^*(cb)^*$	False	False	False	False
$c + cab + (bc)^*$	False	True	True	True	$(aa)^*bb(a)^*$	False	False	False	True
$(c)^*b + b(b)^*$	True	True	True	True	$a(bc)^*$	False	True	True	True
$(ca(c)^*)^*$	False	False	False	False	$(ca)^*(ab)^*$	False	True	True	True
$bab(ba)^*$	False	True	True	True	$((ac)^*)^*$	False	True	True	True
$bbbc(ca)^*$	False	True	True	True	$(ba + a + c)^*$	False	False	False	False
$(abcb)^*$	False	True	True	True	$((a)^*)^*(cc)^*$	False	False	False	True
$cc + cc(bb)^*$	False	False	False	True	$((b)^*a + c)^*$	False	False	False	False
$bcb + (a)^*aa$	True	True	True	True	$(ba)^*cb + aa$	False	True	True	True
$a + c(a)^*cb + a$	True	True	True	True	$(baaa)^*$	False	True	True	True
$(cb(c)^*)^*$	False	False	False	False	$(baa + b)^*$	False	False	False	False
$((b)^*bc)^*$	False	False	False	False	$ccc(b)^*$	True	True	True	True
$(a + aac)^*$	False	False	False	False	$aa(c)^* + acca$	True	True	True	True
$(cb)^*caa + b$	False	True	True	True	$(a)^*caaa + c$	True	True	True	True
$caaba + (a)^*$	True	True	True	True	$((a)^*bb)^*$	False	False	False	False
$(cc)^*bac$	False	False	False	True	$(bb)^*(c)^*ac$	False	False	False	True
$(b + ac + c)^*$	False	False	False	False	$(b)^*b + aa + bba$	True	True	True	True
$(acba)^*$	False	True	True	True	$(b + bc)^*$	False	False	False	False
$cbac(a)^*$	True	True	True	True	$(b)^*cc(b)^*ca$	True	True	True	True
$(a)^*(a)^*(bb)^*$	False	False	False	True	$aca(a)^*$	True	True	True	True
$((a)^*ab)^*$	False	False	False	False	$bcb(bb)^*$	False	False	False	True
$baca(bc)^*$	False	True	True	True	$(a)^*a(b + b)^*$	True	True	True	True
$(ca)^*(cc)^*$	False	False	False	True	$b + c + ac(ac)^*$	False	True	True	True
$(ab + aabb)^*$	False	False	False	False	$(ab + baa)^*$	False	True	True	True
$(aa)^*$	False	False	False	True	$(a^+b^+)^+$	False	False	False	False
$(ab)^+a^+$	False	True	True	True	$(ab)^+a^+b^+$	False	True	True	True
$(ab)^+a^+b^+a^+$	False	True	True	True	$((ab)^+b^+)^+$	False	False	False	False
$((ab)^+b^+)^k$	False	True	True	True	$(ab + ba)^*$	False	True	True	True
$(ab)^+b(ab)^+$	False	True	True	True	$(ab + bba)^*$	False	False	True	True
$(a + b)^+k$	True	True	True	True	abe^*be^*	False	False	False	False
$(ab)^*$	False	True	True	True	$(a(ab)^*b)^*$	False	True	True	True
$b\Sigma^*$	True	True	True	True	Σ^*b	False	False	False	False
$(\Sigma \setminus \{a, b_0\})^*a(\Sigma \setminus \{b_1\})^*$	True	True	True	True	$(\Sigma \setminus \{a_1, b_0\})^*a_1(\Sigma \setminus \{a_2, b_1\})^*a_2(\Sigma \setminus \{b_2\})^*$	True	True	True	True
$\Sigma^*a\Sigma^*$	True	True	True	True	$\Sigma^*ab\Sigma^*$	True	True	True	True
$a^*(ba^*ba^*)^*$	False	False	False	True	$\Sigma^*a\Sigma^*b\Sigma^*$	True	True	True	True
$(\Sigma \setminus b_0)^*a(\Sigma \setminus a, b_1)^*$	False	False	False	False					

Table 1: Set of 125 regular languages used in the experiments in Figure 3 and in Section B.1. For each language, we report membership in R , R^ω , $R \circ G$, and C-RASP (where True in a column denotes membership in the column’s class). For each language, we sampled 10K words for the training set (lengths l_{min} –50) and 1K words for each evaluation length bin.

The syntactic monoid $M(L)$ of a language L is the minimal monoid (up to isomorphism) which recognizes L .

Definition 17 (Basic Units). The three basic semigroup units U_1, U_3, U_2 are given with their multiplication tables, where rows denote the first operand and columns denote the second.

Formal Language	R	C-RASP	R ^ω	R ◦ G
(ab)*bb(aabb)*	False	True	True	True
(abb)*(a)(c+a)(a)cbb+a*(c)	False	True	True	True
(b*ca+ca)(ac+b)(c+a)*(b)*+(bc)*(b+c)(c)+a(c)*caa	False	True	True	True
(b*bc+bb)c(a)(a)c*aa+(bc)*(b)*(b)b*aa+(a+a)(b)aba+(c)*c*bc+b*+bc+a	False	True	True	True
(cba)*a	False	True	True	True
(cac)*b(b+c)*a*+bb+a(b)a*ba+(a)*c*bb	False	True	True	True
(cb)*ab+(aa+a)*a(a)bba	False	True	True	True
(ac+aa+c)(ca+b+b)(c+a)*ba*ab+(ab+c)*(b+c)*(b)*a*ac+c+b*abc+b*cb+bc+c+b	False	True	True	True
(c*cc+b)(bc)*(c+a)*+(bc+a)*+b	False	True	True	True
(acc)*+(ab+c+b)(a+b)*+(a+b)+cab	False	True	True	True
(cac)*(ca+c+c)(c)(b)*ccc+(ac+c+a)*a*(c)*ba+(a+a)*(a)a	False	True	True	True
(bcb)*(b)(b)*(c)*a*	False	True	True	True
(a*ac+ba+a+a)(ab+c)*ba*b*bc	False	True	True	True
(bcb+c+c+c)(bc+c+b)(b+a)*(a)*bba	False	False	False	False
(a*cc+bb+b)(ac)*b*+(c+b+b)*(a)(b)a*ba+(a+c)*cc	False	False	False	False
(bcb+cb+a+c)*	False	False	False	False
(ccb+ac+c+b)+(cc+b)*(b+c)*(c)cba+a*	False	False	False	False
b*+(cb+a)*(c+b)*c*c*c+(a+a)*(c)b*+(b)*	False	False	False	False
(b*c+ba)*c*+(aa+b+c)(a+c)a*ccc	False	False	False	False
(a*ab+cb+a+c)*	False	False	False	False
(cbb+cb+a+a)(ca+b+c)*(c+b)*(b)*b*+(cb)*(a+b)(b)+b(b)c*b	False	False	False	False
(b*ab+ab+a+c)*(cb+b)(b)*+c*(b+b)*(c)*	False	False	False	False
(a*b+b+ba+b+b)*+b*(b+a)(c)acb+c(b)b*b+aaba+cc	False	False	False	False
(c*bb+bb+c)+(ab+c+c)*ab+(b+c)*a+ab	False	False	False	False
(c*+bb+c+a)(ac+a)c(c)a*ab+(cb+b+a)(a)a*c+(c+b)*(b)b*a	False	False	False	False
(b*ca+cb+b)*(a+a)(b+a)a	False	False	False	False
(baa+ab+b)*	False	False	False	False
(c*+ac+a+b)*b*(c+b)	False	False	False	False
(a*cb+ca+b+a)c+a*c*c*+(b+a)*(a)*b*ac+(b)a*c	False	False	False	False
a*+(aa+a+c)(a)(b)*b+(b+a)*(a)bba	False	False	False	False
(abb+b)(ba+c+c)(c+a)+(b)(a+b)*+(c)b*c*+(a)*a*	True	True	True	True
(b*cc+bb+b+a)(ca+b)	True	True	True	True
(b*b+ca+b+c)(a+a+c)*(a+c)+(aa+c)a(a)	True	True	True	True
(a*)*	True	True	True	True
(aca)*(a+c+a)*+(ba+b+a)*cc*c*+(c)(b)*b+b*aab+b	True	True	True	True
(cbc)(bc)a*(c)*a*ab+a*(a+b)(c)c*bb+c*(c)*a*aa	True	True	True	True
b*(ba+c)c*c*c+c(b+b)*bccc+(c)a*c*bb+a*c*cc	True	True	True	True
b+b*(c)(b)cca+ac*baa+(b)*ac+b*cb+b+c+b	True	True	True	True
(acc+c+c+b)(a)(a+a)*(a)*bbb+(ba+a+b)+a*ca*aa+(b)+acc	True	True	True	True
a*b*+a(b+a)(a)b*ac	True	True	True	True
(a*b)*+(ac+b+a)b*bc*cb	False	False	False	False
(a*ac)+c*b(c)*+(c+b)*+(b)aac	True	True	True	True
a*+(a)*	True	True	True	True
(a+bb)*a*	False	False	False	True
(b*bc)+(aa+c)*	False	False	False	True
(cc+aa+a+b)(aa+b+b)*+b*b+(b+b)(c)aca+(a)*acb+cab	False	False	False	True
(b+aa+b+c)*	False	False	False	True
b(aa+c)*(b)(b)cca	False	False	False	True
(a)+(bb)*+a*a*cac	False	False	False	True
(b*cc)*	False	False	False	False

Table 2: Set of 50 regular languages used in the experiments in B.2. For each language, we report membership in **R**, C-RASP, **R^ω** and **R ◦ G** (where True in a column denotes membership in the column’s class). For each language, we sampled 10K words for the training set (lengths $l_{min} - 200$) and 1K words for each evaluation length bin.

U_1	$\begin{array}{c c} 0 & 1 \\ \hline 0 & 0 \\ 1 & 1 \end{array}$	U_3	$\begin{array}{c c} a & b \\ \hline a & b \\ b & a \end{array}$	U_2	$\begin{array}{c cc} 1 & a & b \\ \hline 1 & a & b \\ a & a & b \\ b & a & b \end{array}$
-------	---	-------	---	-------	---

Definition 18 (Basic monoid operations). We define basic monoid operations.

- *Submonoid.* A monoid M is a submonoid of N (usually written $M \leq N$) whenever $M \subseteq N$ and M is closed under the monoid operation of N .
- *Direct product.* The direct product $M \times N$ of monoids M and N has elements (m, n) where $m \in M$ and $n \in N$, with the operation $(m_1, n_1) \cdot_{M \times N} (m_2, n_2) = (m_1 \cdot_M m_2, n_1 \cdot_N n_2)$.
- *Homomorphism.* A homomorphism of monoids $\phi: M \rightarrow N$ is a function such that $\phi(1_M) = 1_N$ and $\phi(m_1)\phi(m_2) = \phi(m_1 m_2)$.
- *Division.* A monoid M divides a monoid N (written $M \preceq N$) whenever there is a submonoid $X \leq N$ of N and a homomorphism $\phi: X \rightarrow M$ such that $M = \phi(X)$.

Definition 19 (Pseudovariety). A pseudovariety of monoids is a class closed under submonoids, division, and finite direct products. A pseudovariety of languages is a class closed under inverse homomorphism, Boolean operations, and shifts ($a^{-1}Lb^{-1}$).

Definition 20. Define $\mathbf{V} \circ \mathbf{W}$ as the pseudovariety generated by all $V \circ W$ where $V \in \mathbf{V}$ and $W \in \mathbf{W}$. Define $\text{wp}^1(\mathbf{V}) = \mathbf{V}$, $\text{wp}^{k+1}(\mathbf{V}) = \text{wp}^k(\mathbf{V}) \circ \mathbf{V}$, and $\text{wpc}(\mathbf{V}) = \bigcup_{k \geq 0} \text{wp}^k(\mathbf{V})$. We also define $\text{wp}^k(M)$ for monoids, taking $\text{wp}^1(M)$ as the pseudovariety generated by M .

We define some pseudovarieties that we use in the paper:

Definition 21. The following are standard, except for $\mathbf{Dy}$.

- $\mathbf{R}$ is the pseudovariety of $\mathcal{R}$ -trivial monoids (Brzozowski & Fich, 1980)
- $\mathbf{A}$ is the pseudovariety of all aperiodic monoids
- $\mathbf{G}$ is the pseudovariety of all finite groups
- $\mathbf{Dy}$ is the pseudovariety generated by $M(\mathcal{D}_k)$ for all $k \in \mathbb{N}$

An important relation on monoid elements we will use is the $\mathcal{R}$ relation (Brzozowski & Fich, 1980):

Definition 22. For $s, t \in M$, we say $s \preceq_{\mathcal{R}} t \Leftrightarrow sM \subseteq tM$.

$\mathcal{R}$ -classes are the equivalence classes for $s \sim_{\mathcal{R}} t \Leftrightarrow [s \preceq_{\mathcal{R}} t \wedge t \preceq_{\mathcal{R}} s]$.

Proposition 23. $M(\mathcal{D}_k) \preceq (U_1 \circ \mathbb{Z}_{k+3}) \circ U_1$, where $\mathbb{Z}_{k+3}$ is the cyclic group of order $k+3$.

Proof. First consider the submonoid of $(U_1 \circ \mathbb{Z}_{k+3}) \times U_1$ generated by $((g, 0), 1)$, $((g, 1), 0)$, and $((g, k+2), 0)$ where $g \in U_1^{\mathbb{Z}_{k+3}}$ given by $g(x) = 0 \iff x \in \{k+1, k+2\}$. It can be verified that $((g, 0), 1) \mapsto \epsilon$, $((g, 1), 0) \mapsto a$, and $((g, k+2), 0) \mapsto b$ extends to a surjective homomorphism $h: U_1 \circ \mathbb{Z}_{k+3} \rightarrow M(\mathcal{D}_k) \times U_1$. Finally, using the fact that direct products divide wreath products, we conclude that $M(\mathcal{D}_k) \preceq (U_1 \circ \mathbb{Z}_{k+3}) \circ U_1$. $\square$

Intuitively, computation in $\mathbb{Z}_{k+3}$ detects if the depth ever exceeds $k+1$, and this depth-violation is detected by U_1 . Another U_1 detects non-emptiness of the string.

D Linear RASP programs

In this section we abstract from C-RASP to the class of *linear RASP programs*, which intuitively are a class of straight-line programs in which each operation at position i can only depend on previously defined operations and positions $j \leq i$. This content follows Thérien & Wilke (2001).

Definition 24 (Linear RASP programs). *Linear RASP programs are those with the syntax*

$$\phi ::= \sigma \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \odot\langle\phi_1, \phi_2, \dots, \phi_k\rangle$$

where $\sigma \in \Sigma$ and $\odot$ is some operator of arity k . For each operator there is an associated collection $K_{\odot} \subseteq \Sigma^* \times (2^{\mathbb{N}})^k$ of words and sets of positions contained in the operator. Semantics are defined

$$\begin{aligned} w, i \models \sigma & \iff w_i = \sigma \\ w, i \models \neg\phi & \iff w, i \not\models \phi \\ w, i \models \phi_1 \wedge \phi_2 & \iff w, i \models \phi_1 \text{ and } w, i \models \phi_2 \\ w, i \models \odot\langle\phi_1, \phi_2, \dots, \phi_k\rangle & \iff (w_{\leq i}, \{j \mid w, j \models \phi_1\}, \dots, \{j \mid w, j \models \phi_k\}) \in K_{\odot} \end{aligned}$$

This is just defining Lindström quantifiers, which can be instantiated by the typical logics as follows.

Example 25. *Linear Temporal Logic is the class of Linear RASP programs with the binary since operator, typically written infix.*

$$K_{\text{since}} = \{(w, R_1, R_2) \mid \exists k \in R_2 \text{ st } k < |w| \text{ and } [k, |w|] \subseteq R_1\}$$

In this way

$$\begin{aligned} w, i \models \phi_1 \text{ since } \phi_2 &\iff (w_{\leq i}, \{j \mid w, j \models \phi_1\}, \{j \mid w, j \models \phi_2\}) \in K_{\text{since}} \\ &\iff \text{there exists } k < i \text{ such that } w, k \models \phi_2 \text{ and } w, j \models \phi_1 \text{ for all } k < j < i \end{aligned}$$

Similarly, C-RASP is the class of linear RASP programs with the operator $(\Lambda, C) = \sum_{1 \leq m \leq k} \lambda_m \cdot \# \phi_m \geq C$

$$K_{(\Lambda, C)} = \left\{ (w, R_1, R_2, \dots, R_k) \mid \sum_{1 \leq m \leq k} \lambda_m \cdot |R_m| \geq C \right\}$$

In this way

$$\begin{aligned} w, i \models \sum_{1 \leq m \leq k} \lambda_m \cdot \# \phi_m \geq C &\iff (w_{\leq i}, \{j \mid w, j \models \phi_1\}, \dots, \{j \mid w, j \models \phi_k\}) \in K_{(\Lambda, C)} \\ &\iff \sum_{1 \leq m \leq k} \lambda_m \cdot |\{j \mid w, j \models \phi_m\}| \geq C \end{aligned}$$

Definition 26. Let Φ and Ψ be classes of linear RASP programs formulas over alphabet Γ and Σ , respectively. Let $G = \{\psi_\gamma\}_{\gamma \in \Gamma}$ be a family of formulas in Ψ . Let θ_G be a mapping of formulas given by

$$\begin{aligned} \theta_G(Q_\gamma) &\mapsto \psi_\gamma \\ \theta_G(\neg \phi) &\mapsto \neg \theta_G(\phi) \\ \theta_G(\phi_1 \wedge \phi_2) &\mapsto \theta_G(\phi_1) \wedge \theta_G(\phi_2) \\ \theta_G(\odot \langle \phi_1, \phi_2, \dots, \phi_k \rangle) &\mapsto \odot \langle \theta_G(\phi_1), \theta_G(\phi_2), \dots, \theta_G(\phi_k) \rangle \end{aligned}$$

We write $\phi[\gamma \mapsto \psi_\gamma]$ for this substitution. This is called a Ψ substitution of Φ formulas. We write $\Phi \star \Psi$ for the class of all Ψ substitutions of Φ formulas. The semantics are as would be expected. By default we let this operation be right-associative.

To help formalize the expressivity of linear RASP programs, we define a class of languages.

Definition 27 (End-Pointed Language). A pointed word is a tuple (w, p) for $w \in \Sigma^*$ and $1 \leq p \leq |w|$. An end-pointed language is a set of pointed words where the point denotes the end of the prefix of the string used for recognition.

- For Φ a class of linear RASP programs, $P(\Phi)$ is the set of pointed languages L for which there exists $\phi \in \Phi$ such that $(w, p) \in L$ iff $w, p \models \phi$.
- For $\mathbf{M}$ a class of typed monoids, $P(\mathbf{M})$ is the set of pointed languages L for which there exists $(M, \mathfrak{T}_M, \mathfrak{E}_M) \in \mathbf{M}$, typed homomorphism $h: \Sigma^* \rightarrow (M, \mathfrak{T}_M, \mathfrak{E}_M)$, a type $\mathfrak{M} \in \mathfrak{T}_M$, and a finite set $C \subseteq M$ such that $(w, p) \in L$ iff $(h(w_1 w_2 \dots w_{p-1}), h(w_p)) \in \mathfrak{M} \times C$.

Programs are classically connected to algebraic characterizations of languages via wreath product principles. We define how to take two classes of programs and obtain a more complex class.

Definition 28 (Program composition). Let Φ, Ψ be classes of linear RASP programs. The class $\Phi \star \Psi$ consists of all programs in Φ where atomic operations may refer to programs in Ψ .

For instance, $\text{C-RASP}_1 \star \text{C-RASP}_1$ is equivalent to the class of depth 2 programs C-RASP_2 . We will see in section E.2 that linear RASP programs can be closely connected to wreath products of monoids.

E Typed Monoids

Krebs (2008) developed a framework for using infinite monoids to recognize languages. We present a restriction of the aforementioned framework to the case of wreath products (a one-sided version of the block product used in previous work), which ultimately provides an exact algebraic characterization of C-RASP. The core issue here is that the wreath product of infinite monoids can generate uncountably many elements, which can be too powerful.

Proposition 29. Consider the classic wreath product $\mathbb{Z} \circ \mathbb{Z}$. Then $M(L) \preceq \mathbb{Z} \circ \mathbb{Z}$ for every L .

Proof. Without loss of generality let $\Sigma = \{0, 1\}$. Consider the submonoid of $\mathbb{Z} \circ \mathbb{Z}$ generated by the image of Σ^* under the homomorphism $\sigma \mapsto (f_\sigma, 1)$ where $f_\sigma(x) = \sigma \cdot 2^{|x|}$. In essence, this creates a mapping $w \mapsto (f_w, |w|)$ where $f_w(0)$ outputs the integer value of the binary number w . Thus, $\mathbb{Z} \circ \mathbb{Z}$ can recognize arbitrary languages. $\square$

This problem motivates the definition of *typed monoids*, which restricts the accepting sets.

E.1 Definitions

Definition 30. A *typed monoid* is a triple $(M, \mathfrak{T}_M, \mathcal{E}_M)$ where M is a finitely generated monoid, $\mathfrak{T}_M$ is a finite Boolean algebra over M , and $\mathcal{E}_M$ is a finite subset of M . Elements of $\mathfrak{T}_M$ are the *types* and elements of $\mathcal{E}_M$ are the *units*. A language L is recognized by $(M, \mathfrak{T}_M, \mathcal{E}_M)$ if there exists a homomorphism $h: \Sigma^* \rightarrow M$ such that $h(\Sigma) \subseteq \mathcal{E}_M$ and $L = h^{-1}(\mathfrak{M})$ for some $\mathfrak{M} \in \mathfrak{T}_M$.

As an example, the language MAJORITY can be recognized by the typed monoid $(\mathbb{Z}, \{(-\infty, 0], [1, \infty), \mathbb{Z}, \emptyset\}, \{-1, 1\})$ via the type $[1, \infty)$ and the homomorphism $a \mapsto 1$ and $b \mapsto -1$. We will typically refer to this typed monoid as $\mathbb{Z}$.

We define morphisms

Definition 31. Let $(S, \mathfrak{T}_S, \mathcal{E}_S)$ and $(T, \mathfrak{T}_T, \mathcal{E}_T)$ be typed monoids. A *typed monoid homomorphism* $h: (S, \mathfrak{T}_S, \mathcal{E}_S) \rightarrow (T, \mathfrak{T}_T, \mathcal{E}_T)$ is a triple $(h_S, h_{\mathfrak{T}_S}, h_{\mathcal{E}_S})$ such that:

- $h_S: S \rightarrow T$ is a monoid homomorphism
- $h_{\mathfrak{T}_S}: \mathfrak{T}_S \rightarrow \mathfrak{T}_T$ is a homomorphism of Boolean algebras
- $\forall \mathfrak{G} \in \mathfrak{T}_S, h_S(\mathfrak{G}) = h_{\mathfrak{T}_S}(\mathfrak{G}) \cap h_S(S)$
- $\forall \mathfrak{J} \in \mathcal{E}_S, h_S(\mathfrak{J}) = h_{\mathcal{E}_S}(\mathfrak{J})$

And due to the compatibility of $h_S, h_{\mathfrak{T}_S}, h_{\mathcal{E}_S}$ we can omit the subscripts. We say that a typed monoid $(S, \mathfrak{T}_S, \mathcal{E}_S)$ recognizes the language $L \subseteq \Sigma^*$ if there is a morphism $h: \Sigma^* \rightarrow S$ with $h(\Sigma) \subseteq \mathcal{E}_S$ and a type $\mathfrak{G} \in \mathfrak{T}_S$ such that $L = h^{-1}(\mathfrak{G})$.

So we want our functions to be compatible with the finite types, which motivates the following definition. In a sense, this requires that all elements of the same type, up to some constant shifting C , behave the same under the function.

Definition 32 (Type-respecting Functions). Let S be a set and $(T, \mathfrak{T}_T, \mathcal{E}_T)$ be a typed monoid and let $C \subseteq T$ be a nonempty finite set of constants. A function $f: T \rightarrow S$ is called *type respecting* with respect to $(T, \mathfrak{T}_T, \mathcal{E}_T)$ and C if the preimage $f^{-1}(s)$ can be described by a finite Boolean combination of conditions of the form $tc \in \mathfrak{T}$ where c is a constant in T (not necessarily in $\mathcal{E}_T$) and $\mathfrak{T} \in \mathfrak{T}_T$.

Intuitively, the image of x under a type-respecting function depends only on the type of xc for some qualified set of constants c . Now the typed wreath product is similar to the untyped case, though the functions are constrained to be type-respecting functions.

Definition 33 (Typed Wreath Product). Let $(M, \mathfrak{T}_M, \mathcal{E}_M), (N, \mathfrak{T}_N, \mathcal{E}_N)$ be two typed monoids, $C \subseteq N$ be a finite set. The *typed wreath product*

$$(U, \mathfrak{T}_U, \mathcal{E}_U) = (M, \mathfrak{T}_M, \mathcal{E}_M) \circ_C (N, \mathfrak{T}_N, \mathcal{E}_N)$$

of $(M, \mathfrak{T}_M, \mathcal{E}_M)$ with $(N, \mathfrak{T}_N, \mathcal{E}_N)$ is defined such that

- $\mathcal{E}_U$ consists of all elements (f, n) , where $n \in \mathcal{E}_N$, and $f: N \rightarrow \mathcal{E}_M$ is a type respecting function (see definition 32) with respect to $(N, \mathfrak{T}_N, \mathcal{E}_N)$ and C
- U is the submonoid of $M \circ N$ generated by $\mathcal{E}_U$
- $\mathfrak{T}_U$ consists of all types $\mathfrak{U}_{\mathfrak{M}, \mathfrak{N}} = \{(f, n) \in U \mid f(1_N) \in \mathfrak{M}, n \in \mathfrak{N}\}$, where $\mathfrak{M} \in \mathfrak{T}_M, \mathfrak{N} \in \mathfrak{T}_N$

Definition 34 (Typed Monoid Pseudovariety). A *typed monoid pseudovariety* is a class of typed monoids closed under

- Division

- *Shifting (changing types by inverse multiplication)*
- *Unit relaxation (swapping out units)*
- *Trivial extension (applying a congruence)*

Definition 35 (Typed Wreath Product Closure). *For typed monoid pseudovarieties define $\mathbf{V} \circ \mathbf{W}$ as the pseudovariety generated by all $V \circ W$ where $V \in \mathbf{V}$ and $W \in \mathbf{W}$. Define $\text{wp}^1(\mathbf{V}) = \mathbf{V}$, $\text{wp}^{k+1}(\mathbf{V}) = \text{wp}^k(\mathbf{V}) \circ \mathbf{V}$, and $\text{wpc}(\mathbf{V}) = \bigcup_{k \geq 0} \text{wp}^k(\mathbf{V})$. We also define $\text{wp}^k((M, \mathfrak{T}_M, \varepsilon_M))$ for typed monoids $(M, \mathfrak{T}_M, \varepsilon_M)$, taking $\text{wp}^1((M, \mathfrak{T}_M, \varepsilon_M))$ as the pseudovariety generated by $(M, \mathfrak{T}_M, \varepsilon_M)$.*

Finally, we note the compatibility of the classical wreath product and the typed wreath product, which will become important in our algebraic decision procedure.

Lemma 36. *Assume $M \preceq S, N \preceq T$ where M, N are finite and S, T are typed. Then*

$$M \circ N \preceq S^N \circ T \quad (1)$$

where $\circ$ is the classical wreath product and $\odot$ is the typed wreath product.

Proof. Call these typed semigroups $(S, \mathfrak{T}_S, \varepsilon_S)$ and $(T, \mathfrak{T}_T, \varepsilon_T)$. Let $h_M: (S', \mathfrak{T}_{S'}, \varepsilon_{S'}) \rightarrow (M, 2^M, M)$ and $h_N: (T', \mathfrak{T}_{T'}, \varepsilon_{T'}) \rightarrow (N, 2^N, N)$ define the divisions. We will define a function $h: (U, \mathfrak{T}_U, \varepsilon_U) \rightarrow M \circ N$ where $(U, \mathfrak{T}_U, \varepsilon_U) \leq (S^N, \mathfrak{T}_{S^N}, \varepsilon_{S^N}) \odot (T, \mathfrak{T}_T, \varepsilon_T)$. First, let U be the submonoid of $(S^N)^T \times T$ generated by (f_d, t) where f_d for $d \in (S')^N$ is defined such that $f_d(1_T) = d$ and $f_d(t) = [n \mapsto d(n + h_N(t))]$ for all $t \in T'$, and $f_d(t) = 1_{(S')^N}$ for $t \in T \setminus T'$. Note that all (f, t) in U satisfy the following constraints:

1. $t \in T'$
2. $\text{Im}(\text{Im}(f)) \subseteq S'$
3. $f(t_1 + t_2)(n) = f(t_1)(n + h_N(t_2))$
4. $f(t_1) = f(t_2)$ whenever $h_N(t_1) = h_N(t_2)$
5. $\text{Im}(f(t)) = 1_{S'^N}$ for $t \in T \setminus T'$

As a submonoid of the wreath product, $\mathfrak{T}_U$ consists of all types $\mathfrak{U}_{\mathfrak{S}\mathfrak{N}, \mathfrak{T}} = \{(f, t) \in U \mid f(1_T) \in \mathfrak{S}\mathfrak{N}, t \in \mathfrak{T}\}$, where $\mathfrak{S}\mathfrak{N} \in \mathfrak{T}_{S'^N}$, $\mathfrak{T} \in \mathfrak{T}_{T'}$ and 1_T is the neutral element of T . Let ε_U consist of (f, t) where $t \in \varepsilon_{T'}$. Define a function $h: (U, \mathfrak{T}_U, \varepsilon_U) \rightarrow M \circ N$ such that $h((f, t)) = (g, h_N(t))$ where $g(n) = h_M(f(1_T)(n))$.

- All functions in U are type-respecting. This is because each of the generators f_d is type-respecting – since h_N is a homomorphism on types $h_N: \mathfrak{T}_T \mapsto 2^N$, the image $f_d(t)$ is determined by a boolean combination of conditions on the type of t .
- h is a surjection, because any $(g, n) \in M \circ N$:
 - There is $f \in U$ such that $g(\cdot) = h_M(f(t)(\cdot))$, because all $d \in (S')^N$ are represented in U . We just pick d to be compatible with g and f_d witnesses the preimage. As for which d to choose, we construct d where for $n_0 \in N$, we pick an element $s_0 \in h_M^{-1}(g(n_0))$ (which exists by the surjectivity of h_M) and set d such that $d(n') = s_0$ for all n' where $g(n') = g(n_0)$.
 - There is $t \in T'$ such that $h_N(t') = n$ by the surjectivity of h_N .

- h is a homomorphism on elements. From (3) we get that $f(t_1 + t_2)(n) = f(t_1)(n + h_N(t_2))$.

$$\begin{aligned}
& h((f_1, t_1)(f_2, t_2)) \\
&= h((f_1 + {}^{t_1}f_2, t_1 t_2)) \\
&= (n \mapsto h_M((f_1 + {}^{t_1}f_2)(1_T)(n)), h_N(t_1 t_2)) \\
&= ((n \mapsto h_M(f_1(1_T)(n))) + (n \mapsto h_M({}^{t_1}f_2(1_T)(n))), h_N(t_1 t_2)) \\
&= ((n \mapsto h_M(f_1(1_T)(n))) + (n \mapsto h_M(f_2(1_T + t_1)(n))), h_N(t_1 t_2)) \\
&= ((n \mapsto h_M(f_1(1_T)(n))) + (n \mapsto h_M(f_2(1_T)(n + h_N(t_1)))), h_N(t_1)h_N(t_2)) \quad \text{by (3)} \\
&= ((n \mapsto h_M(f_1(1_T)(n))) + {}^{h_N(t_1)}(n \mapsto h_M(f_2(1_T)(n))), h_N(t_1)h_N(t_2)) \\
&= (n \mapsto h_M(f_1(1_T)(n)), h_N(t_1))(n \mapsto h_M(f_2(1_T)(n)), h_N(t_2)) \\
&= h((f_1, t_1))h((f_2, t_2))
\end{aligned}$$

- h is a homomorphism on types. Let $\mathcal{U}_{\mathfrak{S}\mathfrak{N}, \mathfrak{T}}$ be a type of $(U, \mathfrak{T}_U, \mathfrak{E}_U)$. Here, $h(\mathfrak{S}\mathfrak{N}) \in 2^{M^N}$ and $h(\mathfrak{T}) \in 2^N$, because h_M, h_N are homomorphisms on types by assumption³. First, h respects complements

$$\begin{aligned}
h(\overline{\mathcal{U}_{\mathfrak{S}\mathfrak{N}, \mathfrak{T}}}) &= h(\{(f, t) \in U \mid f(1_T) \notin \mathfrak{S}\mathfrak{N} \text{ or } t \notin \mathfrak{T}\}) \\
&= \{h((f, t)) \in M \circ N \mid h(f(1_T)) \notin h(\mathfrak{S}\mathfrak{N}) \text{ or } h(t) \notin h(\mathfrak{T})\} \\
&= 2^{M^N} \times 2^N \setminus \{h((f, t)) \in M \circ N \mid h(f(1_T)) \in h(\mathfrak{S}\mathfrak{N}), h(t) \in h(\mathfrak{T})\} \\
&= \overline{h(\mathcal{U}_{\mathfrak{S}\mathfrak{N}, \mathfrak{T}})}
\end{aligned}$$

And h respects union

$$\begin{aligned}
& h(\mathcal{U}_{\mathfrak{S}\mathfrak{N}_1, \mathfrak{T}_1} \cup \mathcal{U}_{\mathfrak{S}\mathfrak{N}_2, \mathfrak{T}_2}) \\
&= h(\{(f, t) \in U \mid f(1_T) \in \mathfrak{S}\mathfrak{N}_1, t \in \mathfrak{T}_1\} \cup \{(f, t) \in U \mid f(1_T) \in \mathfrak{S}\mathfrak{N}_2, t \in \mathfrak{T}_2\}) \\
&= \{h((f, t)) \in U \mid f(1_T) \in \mathfrak{S}\mathfrak{N}_1, t \in \mathfrak{T}_1\} \cup \{h((f, t)) \in U \mid f(1_T) \in \mathfrak{S}\mathfrak{N}_2, t \in \mathfrak{T}_2\} \\
&= h(\{(f, t) \in U \mid f(1_T) \in \mathfrak{S}\mathfrak{N}_1, t \in \mathfrak{T}_1\}) \cup h(\{(f, t) \in U \mid f(1_T) \in \mathfrak{S}\mathfrak{N}_2, t \in \mathfrak{T}_2\}) \\
&= h(\mathcal{U}_{\mathfrak{S}\mathfrak{N}_1, \mathfrak{T}_1}) \cup h(\mathcal{U}_{\mathfrak{S}\mathfrak{N}_2, \mathfrak{T}_2})
\end{aligned}$$

And h preserves the inf and sup

$$\begin{aligned}
h(\emptyset) &= \emptyset \\
h(U) &= \{h(f_d, t) \mid d \in (S')^N, t \in T'\} \\
&= \{(h_M(d), h_N(t)) \mid d \in (S')^N, t \in T'\} \\
&= M^N \times N
\end{aligned}$$

□

E.2 Proof of Typed Wreath Product Principle

First, we show how the type of an element is computed within the wreath product. Here we write π_1 and π_2 as the projections from the first and second coordinates of $M^N \times N$

Lemma 37. *Let $h: \Sigma^* \rightarrow (T, \mathfrak{T}_T, \mathcal{J}) = (M, \mathfrak{T}_M, \mathfrak{E}_M) \circ_C (N, \mathfrak{T}_N, \mathfrak{E}_N)$ for $C \subseteq N$. Let $\pi_1(\mathfrak{T}) \in \mathfrak{T}_M$ be such that $t \in \mathfrak{T} \iff \pi_1(t)(1_N) \in \pi_1(\mathfrak{T})$. Then $h(w) \in \mathfrak{T} \in \mathfrak{T}_T$ iff*

³Note that Krebs (2008) defined the type of (f, t) independent of t (in order to ease the connection to logic), in which case the image under h would not be a type in $2^{M^N} \times 2^N$. By conditioning the type on t , the homomorphism on types goes through without need for additional direct products to enforce the types of T .

$$\sum_{1 \leq i \leq |w|} \pi_1(h(w_i)) \left(\prod_{1 \leq j < i} \pi_2(h(w_j)) \right) \in \pi_1(\mathfrak{T}) \in \mathfrak{T}_M$$

$$\prod_{1 \leq j < \leq i} \pi_2(h(w_j)) \in \pi_2(\mathfrak{T}) \in \mathfrak{T}_N$$

Proof. We walk through the computation. First, we note that $h(w) \in \mathfrak{T}$ iff $\pi_1(h(w)) \in \pi_1(\mathfrak{T})$, and this type $\pi_1(\mathfrak{T})$ exists by the definition of the typed wreath product. Let $h(w_i) = (f_i, n_i)$ where $f_i \in M^N$ and $n_i \in N$. Then compute

$$\begin{aligned} h(w) &= h(w_1)h(w_2) \dots h(w_{|w|}) \\ &= (f_1, n_1)(f_2, n_2) \dots (f_{|w|}, n_{|w|}) \\ &= \left(f_1 + {}^{n_1}f_2 + \dots + ({}^{\prod_{1 \leq j \leq |w|-1} n_j})f_{|w|}, \prod_{1 \leq i \leq |w|} n_i \right) \\ &= \left(\sum_{1 \leq i \leq |w|} ({}^{\prod_{1 \leq j < i} n_j})f_i, \prod_{1 \leq i \leq |w|} n_i \right) \end{aligned}$$

Recall by the definition of the wreath product that ${}^nf(x) = f(xn)$, so ${}^nf(1_N) = f(n)$.

$$\begin{aligned} \pi_1(h(w))(1_N) \in \pi_1(\mathfrak{T}) &\iff \sum_{1 \leq i \leq |w|} ({}^{\prod_{1 \leq j < i} n_j})f_i(1_N) \in \pi_1(\mathfrak{T}) \\ &\iff \sum_{1 \leq i \leq |w|} f_i \left(\prod_{1 \leq j < i} n_j \right) \in \pi_1(\mathfrak{T}) \\ &\iff \sum_{1 \leq i \leq |w|} \pi_1(h(w_i)) \left(\prod_{1 \leq j < i} \pi_2(h(w_j)) \right) \in \pi_1(\mathfrak{T}) \end{aligned}$$

The computation of the second coordinate is routine as is computed in $(N, \mathfrak{T}_N, \mathcal{E}_N)$

$$\pi_2(h(w)) \in \pi_2(\mathfrak{T}) \iff \prod_{1 \leq j < \leq i} \pi_2(h(w_j)) \in \pi_2(\mathfrak{T}).$$

□

Now connection between the algebraic and logical formulations is often spelled out using statements in the form of a “wreath product principle”.

Theorem 38 (Typed Wreath Product Principle). *Let Φ, Ψ be classes of linear RASP programs and $\mathbf{M}, \mathbf{N}$ be pseudovarieties of monoids such that $L(\Phi) = L(\mathbf{M})$ and $P(\Psi) = P(\mathbf{N})$. Then $L(\Phi \star \Psi) = L(\mathbf{M} \circ \mathbf{N})$.*

Proof.

- Suppose $h: \Sigma^* \rightarrow (T, \mathfrak{T}_T, \mathcal{J}) = (M, \mathfrak{T}_M, \mathcal{E}_M) \circ_C (N, \mathfrak{T}_N, \mathcal{E}_N)$ for $C_N \subseteq N$. For each type $\mathfrak{T}_{\mathfrak{M}, \mathfrak{N}} \in \mathfrak{T}_T$ we will construct a formula $\theta_{\mathfrak{T}} \in \Phi \star \Psi$ such that $w \models \theta_{\mathfrak{T}} \iff h(w) \in \mathfrak{T}$. Observe from Lemma 37 that $h(w) \in \mathfrak{T}$ iff

$$\sum_{1 \leq i \leq |w|} \pi_1(h(w_i)) \left(\prod_{1 \leq j < i} \pi_2(h(w_j)) \right) \in \mathfrak{M} \quad \text{and} \quad \prod_{1 \leq j < \leq i} \pi_2(h(w_j)) \in \mathfrak{N}.$$

The first coordinate can be viewed as recognition of a language by $\mathfrak{M}$ in M via a homomorphism $h_1: (\mathfrak{T}_N^C)^* \rightarrow (M, \mathfrak{T}_M, \mathfrak{E}_M)$. By assumption obtain a $\phi \in \Phi$ that recognizes this language. As for the word $w' \in (\mathfrak{T}_N^C)^*$ we have that

$$w'_i = (\mathfrak{N}_{c_1}, \mathfrak{N}_{c_2}, \dots, \mathfrak{N}_{c_{|C|}}) \iff \bigwedge_{c \in C} \left[\left(c \prod_{1 \leq j < i} \pi_2(h(w_j)) \right) \in \mathfrak{N}_c \right]$$

Observe that for each c , we use $(N, \mathfrak{T}_N, \mathfrak{E}_N)$ to recognize an end-pointed language accepting with type $\mathfrak{N}_c$. Then by assumption we obtain $\psi_c \in \Psi$ recognizing each prefix, and then use the substitution

$$\phi_{\mathfrak{M}} \left[(\mathfrak{N}_{c_1}, \mathfrak{N}_{c_2}, \dots, \mathfrak{N}_{c_{|C|}}) \mapsto \bigwedge_{c \in C} \psi_c \right]$$

The second coordinate can be viewed as recognition of a language by $\mathfrak{N}$ in N . Since $\mathbf{N}$ is a pseudovariety, $L(\mathbf{N}) \subseteq P(\mathbf{N})$. We obtain a formula $\psi_{\mathfrak{N}} \in \Psi$ such that recognizes the same language and define $\theta_{\mathfrak{T}} = \phi_{\mathfrak{M}} \wedge \psi_{\mathfrak{N}}$. This results in a formula $\theta_{\mathfrak{T}} \in \Phi \star \Psi$ which recognizes the same language as $(T, \mathfrak{T}_T, \mathcal{T})$.

- The other direction is similar. Suppose we have a formula with substitution $\phi[\gamma \mapsto \psi_\gamma] \in \Phi \star \Psi$. Because $P(\Psi) = P(\mathbf{N})$, there are typed monoids N_γ that can compute the substitution's output γ at each position. That is, for each γ there is a type $\mathfrak{N}_\gamma$ and homomorphism $h: \Sigma^* \rightarrow N_\gamma$

$$w, i \models \psi_\gamma \iff h(w_{<i}) \in \mathfrak{N}_\gamma$$

Using this, we can compute the substitution into a word over Γ . Then because $L(\Phi) = L(\mathbf{M})$, we obtain a typed monoid M that can recognize the resulting language over Γ . The entire computation can thus be computed using a monoid in $\mathbf{M} \circ \mathbf{N}$.

□

F C-RASP

Here we reiterate some of the definitions of C-RASP, which can be found in prior papers by [Yang & Chiang \(2024\)](#); [Huang et al. \(2025\)](#).

F.1 Definitions

Definition 39. *The syntax of C-RASP formulas is defined:*

$$\phi ::= \sigma \mid \neg \phi_1 \mid \phi_1 \wedge \phi_2 \mid \sum_{t \in \mathcal{T}} \alpha_t \cdot \overset{\leftarrow}{\#}[\phi_t] \sim k$$

where $\sigma \in \Sigma$, $\alpha_i, k \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \geq, >\}$. The semantics of formulas is defined as follows:

$$\begin{aligned} w, i \models \sigma & \iff w_i = \sigma \\ w, i \models \neg \phi & \iff w, i \not\models \phi \\ w, i \models \phi_1 \wedge \phi_2 & \iff w, i \models \phi_1 \text{ and } w, i \models \phi_2 \\ w, i \models \sum_{t \in \mathcal{T}} \alpha_t t \sim k & \iff \sum_{t \in \mathcal{T}} \alpha_t \cdot |\{j \in [1, i] \mid w, j \models \phi_t\}| \sim k. \end{aligned}$$

We write $w \models \phi$ iff $w \langle \text{EOS} \rangle, |w| + 1 \models \phi$ where $\langle \text{EOS} \rangle \notin \Sigma$ is a special end-of-sequence symbol. and we say that ϕ defines the language $L(\phi) = \{w \mid w \models \phi\}$.

In the sequel, we will use a DAG (directed acyclic graph) representation of C-RASP formulas, where a subformula ϕ may be used *multiple times* in a formula. Such a formula can be thought of as a straight-line *program*, i.e., a sequence $\phi = (\phi_i)_{i=1}^n$, where ϕ_i is any C-RASP definition that could refer to ϕ_j with $j < i$.

Definition 40. The syntax of C-RASP is as follows:

$$\begin{aligned} \phi &::= \sigma \mid t_1 < t_2 \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 & \sigma \in \Sigma & \text{Boolean-valued formulas} \\ t &::= \#[\phi_1] \mid t_1 + t_2 \mid 1 & & \text{integer-valued terms} \end{aligned}$$

The semantics of formulas is defined as follows:

$$w, i \models \sigma \iff w_i = \sigma \quad (3a)$$

$$w, i \models \neg\phi \iff w, i \not\models \phi \quad (3b)$$

$$w, i \models \phi_1 \wedge \phi_2 \iff w, i \models \phi_1 \text{ and } w, i \models \phi_2 \quad (3c)$$

$$w, i \models t_1 < t_2 \iff t_1^{w,i} < t_2^{w,i}. \quad (3d)$$

The semantics of terms is defined as follows:

$$\#[\phi]^{w,i} = |\{j \in [1, i] \mid w, j \models \phi\}| \quad (4a)$$

$$(t_1 + t_2)^{w,i} = t_1^{w,i} + t_2^{w,i} \quad (4b)$$

$$1^{w,i} = 1. \quad (4c)$$

We write $w \models \phi$ if $w, |w| \models \phi$, and we say that ϕ defines the language $L(\phi) = \{w \mid w \models \phi\}$.

The table below shows how this program works for the string $((()))()$, which belongs to $\mathcal{D}_2$.

predicate	definition	description	a	a	b	b	a	b
a		is left paren	T	T	⊥	⊥	T	⊥
b		is right paren	⊥	⊥	T	T	⊥	T
C_a	$:= \#[\underline{a}]$	num of left parens	1	2	2	2	3	3
C_b	$:= \#[\underline{b}]$	num of left parens	0	0	1	2	2	3
ϕ_{low}	$:= C_a - C_b \geq 0$	depth above 0	T	T	T	T	T	T
ϕ_{up}	$:= C_a - C_b \leq 2$	depth below 2	T	T	T	T	T	T
ϕ_{bounded}	$:= \phi_{\text{low}} \wedge \phi_{\text{up}}$	depth bounded	T	T	T	T	T	T
ϕ_{matched}	$:= \#[\neg\phi_{\text{bounded}}] = 0$	depth bounded everywhere	T	T	T	T	T	T
ϕ_{balanced}	$:= C_a = C_b$	balanced at end	⊥	⊥	⊥	T	⊥	T
$\phi_{\mathcal{D}_2}$	$:= \phi_{\text{matched}} \wedge \phi_{\text{balanced}}$	acceptance	⊥	⊥	⊥	T	⊥	T

The table below shows how this program works for the string $()()()$, which does not belong to $\mathcal{D}_2$.

predicate	definition	description	a	b	b	a	b	a
a		is left paren	T	⊥	⊥	T	T	⊥
b		is right paren	⊥	T	T	⊥	T	⊥
C_a	$:= \#[\underline{a}]$	num of left parens	1	1	1	2	2	3
C_b	$:= \#[\underline{b}]$	num of left parens	0	1	2	2	3	3
ϕ_{low}	$:= C_a - C_b \geq 0$	depth above 0	T	T	⊥	T	⊥	T
ϕ_{up}	$:= C_a - C_b \leq 2$	depth below 2	T	T	T	T	T	T
ϕ_{bounded}	$:= \phi_{\text{low}} \wedge \phi_{\text{up}}$	depth bounded	T	T	⊥	T	⊥	T
ϕ_{matched}	$:= \#[\neg\phi_{\text{bounded}}] = 0$	depth bounded everywhere	T	⊥	⊥	⊥	⊥	⊥
ϕ_{balanced}	$:= C_a = C_b$	balanced at end	⊥	⊥	⊥	T	⊥	T
$\phi_{\mathcal{D}_2}$	$:= \phi_{\text{matched}} \wedge \phi_{\text{balanced}}$	acceptance	⊥	⊥	⊥	⊥	⊥	⊥

F.2 C-RASP as a pseudovariety of Languages

Proposition 41. C-RASP defines a pseudovariety of languages.

Proof. We sketch the proof here.

- Closed under Boolean combinations: by definition.
- Closed under inverse homomorphisms: follow the construction in [Yang et al. \(2025, Lemma F.2\)](#) or [Huang et al. \(2025, Lemma 34\)](#), but ignoring positional predicates and allowing arbitrary symbols in the image of the homomorphism.

- Closed under factors: Let L be in C-RASP, and $a \in \Sigma$. To obtain $a^{-1}L$, we can detect the beginning in C-RASP and simulate the computations at a prefix a . To obtain La^{-1} , at $\langle \text{EOS} \rangle$ we simulate the computations at $a\langle \text{EOS} \rangle$.

□

Because the languages form a pseudovariety, there must exist a corresponding class of monoids (Behle et al., 2011, Theorem 2).

Remark 42. *If we defined recognition without the $\langle \text{EOS} \rangle$ symbol, the result would not be a pseudovariety, because of the special role played by the final position. For instance, a C-RASP program can separate $\{a, b\}^*b$ from $\{a, b\}^*a$, but taking the inverse of a homomorphism that deletes e and keeps a, b unchanged leads to the two sets $\{a, b\}^*be^*$ from $\{a, b, e\}^*ae^*$ which no C-RASP program can separate.*

F.3 Existing Characterizations

Previous work has explored upper and lower bounds for the regular languages of C-RASP and the related logic $\widehat{\text{MAJ}}_2[<]$, though none have arrived at an exact characterization. Here, we summarize a few of these previously known results.

Example 43. *We know the following language characterizations*

1. $\mathbf{R} \subset \text{C-RASP} \cap \mathbf{REG}$, because any $\mathcal{R}$ -trivial language is definable using existential quantification to the left, which is implementable in C-RASP ($\#[\phi] \geq 1$).
2. $\text{C-RASP} \cap \mathbf{REG} \subset \mathbf{A}$, since any periodic regular language like $(aa)^*$ is not definable in C-RASP (Huang et al., 2025, Lemma 38).
3. Σ^*b , is not in C-RASP (Huang et al., 2025, Lemma 38).
4. $\Sigma^*bb\Sigma^*$ is not in C-RASP, because it is not in the larger class $\widehat{\text{MAJ}}_2[<]$ by Lemma 6.11 in Krebs (2008)
5. For any k , $(a^+b^+)^k$ in C-RASP by Yang et al. (2025)
6. $(ab)^+$ in C-RASP

The next section develops our exact characterization.

F.4 Algebraic Characterization of C-RASP

Theorem 11. $L \in \text{C-RASP} \iff M(L) \in \text{wpc}(\mathbb{Z})$

Proof. We will show that $L(\text{C-RASP}) = L(\text{wpc}(\mathbb{Z}))$, which will be equivalent to the theorem statement. First, we note that $\text{wpc}(\mathbb{Z})$ and $\text{wp}^2(\mathbb{Z})$ are pseudovarieties of typed monoids by definition. Then, we establish that $P(\text{C-RASP}_1) = P(\text{wp}^2(\mathbb{Z}))$, noting that C-RASP is an instance of a linear RASP program. First, by Yang et al. (2025, Lemma A.4) every C-RASP program can be written such that it only counts over positions $j < i$, with Boolean operations testing the symbol at i . The strict counting can be translated into a typed monoid over $\text{wp}^2(\mathbb{Z})$ (since the equations in C-RASP may not just be $\# \phi \geq 0$, we require another wreath product to add constants into the equation), and the testing of the symbol at i can be handled by constants in $\text{wp}^2(\mathbb{Z})$. The other direction is similar – for every type of $\mathbb{Z}$, there exists a C-RASP program that checks if the running sum is in that type. Thus by section E.2, $L(\text{C-RASP}_1) = L(\mathbb{Z} \circ \mathbb{Z})$. From this base case we can build up to $L(\text{C-RASP}_k) = L(\text{wp}^{2k}(\mathbb{Z}))$ by induction, and thus $L(\text{C-RASP}) = \bigcup_{k \geq 0} L(\text{C-RASP}_k) = \bigcup_{k \geq 0} L(\text{wpc}^k(\mathbb{Z})) = \text{wpc}(\mathbb{Z})$. □

G Derived Categories

Here we provide a formal exposition of the ideas that were informally presented in the body of the paper. The notions are based on Tilson (1987) but with some notational adaptations;

we refer to that paper for full formal definition, and for proofs of well-definedness. As mentioned above, we will principally be interested in finite categories for use as algebraic objects.

Definition 44 (Category). *A category X consists of a set of objects $\text{Obj}(X)$ and for each $c, c' \in \text{Obj}(X)$ a homset of arrows $X(c, c')$, often written as $x: c \rightarrow c'$ for $x \in X(c, c')$. A category is endowed with the following algebraic structure:*

- For arrows we have an associative composition operation, where $s: c \rightarrow c'$ and $t: c' \rightarrow c''$, compose into an arrow $st: c \rightarrow c''$. Furthermore, for $s: c_1 \rightarrow c_2$, $t: c_2 \rightarrow c_3$, and $v: c_3 \rightarrow c_4$, we have that $(st)v = s(tv)$.
- For objects c we have an identity arrow $1_c: c \rightarrow c$, where $s1_c = s$ and $1_{c'}t = t$ for $s: c' \rightarrow c$ and $t: c \rightarrow c''$.

Remark 45. When the ambient category C is unambiguous, we also write $\text{Hom}(x \rightarrow x')$ for $C(x, x')$.

We will think of a monoid as a single-object category with the monoid elements as arrows of the category. Between categories we can define relations, which do not necessarily have to be functions.

Definition 46 (Category relation). *Let X, Y be categories. A category relation $f: X \rightarrow Y$ has*

- An object relation $f: \text{Obj}(X) \rightarrow \text{Obj}(Y)$, thought of as a subset of $X \times Y$.
- For any corresponding edge sets $X(c, c')$ and $Y(d, d')$ where $d \in cf, d' \in c'f$, an edge set relation $f: X(c, c') \rightarrow Y(d, d')$.

and satisfies the property that $\#f$, defined as follows, is a subcategory $\#f$ of $X \times Y$:

- $\text{Obj}(\#f) = \{(c, d): d \in cf\}$
- $\#f[(c, d), (c', d')] = \{(x, y) \mid x \in X(c, c'), y \in Y(d, d'), y \in xf\}$

The most important kind of relation for us will be a relational morphism, denoted $\phi: X \triangleleft Y$. In a sense, this is a generalization of a homomorphism where you can take any function on the atomic elements of the X as a generator of the resulting relation $X \triangleleft Y$ after we close under composition in X .

Definition 47 (Relational Morphism). *A relational morphism $f: C \triangleleft C'$ between categories C and C' is a category relation where the object relation is a function and each hom-set relation is fully-defined. A relational morphism where the hom-set relations are injective is called a division.*

Remark 48. It is convenient to view relational morphisms as set-valued functions. That is, if $f: C \triangleleft C'$ and $\alpha \in C(x, x')$ is an arrow in C , then we write $f(\alpha)$ for the set $\{\beta \in C'(y, y') : (\alpha, \beta) \in \#f[(x, x'), (y, y')]\}$, where $(x, y), (x', y') \in \text{Obj}(\#f)$.

The condition that $\#f$ be a category implies in particular

$$f(\alpha)f(\beta) \subseteq f(\alpha\beta) \quad (5)$$

whenever $\alpha \in C(x, x'), \beta \in C(x', x'')$. It also implies that the image of an identity arrow at some object of C always includes the identity arrow at the corresponding target object in C' .

The notion of homomorphism and division for finite monoids is a special case of the definitions above for categories. Then, as discussed above, the analogue of the kernel of a group homomorphism (and thus the analogue of a “divisor”) for monoids is the *derived category*. The worked example in section 4.2 hopefully provides some intuition on the structure of the derived category.

Definition 49 (Derived Category). *Let $\phi: M \triangleleft N$ be a relational morphism between monoids. The derived category D_ϕ of the relational morphism is defined with $\text{Obj}(D_\phi) = \phi(M)$ and $\text{Hom}(n_1, n_2) = \{n_1 \rightarrow_{(m,n)} : \phi^{-1}(n_1) \rightarrow \phi^{-1}(n_2) \mid (m, n) \in \#f, n_1 n = n_2\}$, where $n_1 \rightarrow_{(m,n)}$ is a function $\phi^{-1}(n_1) \rightarrow \phi^{-1}(n_1 n)$ mapping any $x \in \phi^{-1}(n_1)$ to $xm \in \phi^{-1}(n_1 n)$. Composition of arrows is given by $(n_0 \rightarrow_{(m_1, n_1)}) (n_1 \rightarrow_{(m_2, n_2)}) = n_0 \rightarrow_{(m_1 m_2, n_1 n_2)}$.*

Remark 50. We note that, as the arrows denote functions on subsets of M , it is possible for $n_1 \rightarrow_{(m,n)}$ and $n_1 \rightarrow_{(m',n)}$ to be identical even if $m \neq m'$, provided m and m' act identically on $\phi^{-1}(n_1)$.

Where helpful for notational clarity, we explicitly include the end object, writing $n_1 \rightarrow_{(m,n)}$ as $n_1 \rightarrow_{(m,n)} n_2$ where $n_2 = n_1 n$.

For readers familiar with the corresponding construction for groups, we provide some intuition. In some sense, $\ker \phi$ for $\phi: G \rightarrow H$ records what information is lost when compressing G into H . By recording what elements collapse into 1_H , we can reconstruct how every other component of G collapses into H (taking advantage of the inverses in the group to form connections between elements). Then, taking the quotient $G/(\ker \phi)$ precisely records what information is lost by this collapse, and by enriching H with this information again we reconstruct G via the division $G \preceq (\ker \phi) \circ H$.

In the case of monoids, we lack the nice closure properties of groups, and thus $\ker \phi$ cannot be used to reconstruct the collapsing behavior of every other component of G . So the corresponding structure must be enriched with additional information. Indeed, in the derived category construction, for each element of $n \in N$ we consider subsets of M which collapse into n via a relational morphism, and then must record specific information about the interactions of elements within and between these subsets. The derived category just stores the essential amount of information in order to reconstruct M via a wreath product $M \preceq V \circ N$, formalized by the Derived Category Theorem (Tilson, 1987):

Theorem 51 (Derived Category Theorem).

1. Let $\phi: M \triangleleft N$ be a relational morphism of monoids, and let V be a monoid satisfying $D_\phi \preceq V$. Then there is a division of monoids $\theta: M \preceq V \circ N$.
2. Let $\theta: M \preceq V \circ N$ be a division of monoids, and let $\phi = \theta\pi: M \triangleleft N$ be the associated relational morphism. Then $D_\phi \preceq V^N$.

H Algebraic Decision Procedure

H.1 Relevant Lemmas

Lemma 52. If $M \preceq S$ and $N \preceq T$ then $M \circ N \preceq S \circ T$.

Proof. This is a standard fact which we sketch out here. For some $S' \leq S$ and $T' \leq T$ there exists surjections $h_M: S' \rightarrow M$ and $h_N: T' \rightarrow N$. Let G' be the subset of $S^T \times T$ generated by all (f, t') where $t' \in T'$, $\text{Im}(f) \subseteq S'$, and $f(t) = 1_S$ for $t \notin T'$, with the wreath product action inherited from $S \circ T$. Define $h: G \rightarrow M \circ N$ by $h(f, t') = (f', h_N(t'))$ where $f'(n) = h_M(f'(t))$ for some $t \in h_N^{-1}(n)$ (the choice does not matter). It is clear that h is a surjective homomorphism via inheritance from h_M and h_N . $\square$

H.2 Bounded-Depth Dyck Monoids

To make constructions computable despite the infinity of $\mathbb{Z}$, we use bounded-depth Dyck monoids $\mathcal{D}_k$ as partial stand-ins for $\mathbb{Z}$. The non- $\perp$ elements of the monoid $\mathcal{D}_k$ can be canonically represented as tuples $(h, h_\downarrow, h_\uparrow)$ where, for a word $w \in \{a, b\}^*$, the syntactic morphism η maps it to

$$h = \sum_{i=1}^{|w|} \phi(w_i)$$

$$h_\uparrow = \max_j \sum_{i=1}^j \phi(w_i)$$

$$h_\downarrow = \min_j \sum_{i=1}^j \phi(w_i)$$

under $\phi(a) = 1, \phi(b) = -1$, provided these numbers are all in $[-k, \dots, k]$; otherwise the word is mapped to $\perp$. The monoid $\mathcal{D}_k$ has a natural action on the set

$$S_k := [-k, \dots, k] \cup \{\perp\} \quad (6)$$

given by

$$j \cdot (h, h_\uparrow, h_\downarrow) = \begin{cases} \perp & \text{if } j = \perp \\ h + j & \text{if } j + h_\downarrow \geq -k \wedge j + h_\uparrow \leq k \\ \perp & \text{else} \end{cases} \quad (7)$$

Formally, the pair $(\mathcal{D}_k, S_k)$ is a *transformation monoid*.⁴ We could develop the decidability proof in terms of wreath products of $(\mathcal{D}_k, S_k)$, which are somewhat different from wreath products of $\mathcal{D}_k$; this would have some advantages because the set S_k naturally behaves as a truncated version of the infinite set $\mathbb{Z}$. However, to avoid introducing more technical notions, we stay on the level of monoids, at the cost of factoring out an extra congruence out of the derived category. Specifically, there is a natural right-congruence on $\mathcal{D}_k$, namely $n \sim m \Leftrightarrow 0n = 0m$. It is a right-congruence in the sense that $0n = 0m \Rightarrow 0nr = 0mr$ for any $n, m, r \in \mathcal{D}_k$. When considering derived categories for relational morphisms to $\mathcal{D}_k$, we will factor this right congruence out of the object set; this is a well-defined construction resulting in a category because $\sim$ is a right-congruence. The result resembles the derived category, but has a more coarse-grained object set consisting of the equivalence classes of $\sim$.

H.3 Decidability Proof (via Finite Derived Categories)

The following formalizes a basic construction in the theory of derived categories; it permits going from a derived category to a wreath product decomposition:

Definition 53 (Extension). *Let M, S, T be finite monoids. Given relational morphisms $\phi : M \triangleleft T$, $\phi' : D_\phi \triangleleft S$, we define $\text{Ext}(\phi', \phi) : M \triangleleft (S \circ T)$ as*

$$\text{Ext}(\phi', \phi)(m) = \{(f, t) \in S^T \times T : t \in \phi(m); \forall x \in \text{Obj}(D_\phi) : f(x) \in \phi'(x \rightarrow_{(m,t)} xt)\}$$

Remark 54. *This construction is made at the top of p. 116, case (a) of the proof of Theorem 5.2 of Tilson (1987). There, it is also proven that it is a relational morphism. It is also proven that if ϕ' is a division, then so is $\text{Ext}(\phi', \phi)$.*

Remark 55. *We write $\text{Ext}(\phi_1, \phi_2, \dots, \phi_n)$ for $\text{Ext}(\phi_n, \text{Ext}(\phi_{n-1}, \dots))$.*

The following is shown in Tilson (1987):

Proposition 56. *$\text{Ext}(\phi', \phi)$ is a well-defined relational morphism $M \triangleleft S \circ T$.*

In the sequel we will write $\mathbb{Z}$ or $(\mathbb{Z}, \mathbb{Z}_+)$ for the typed monoid $(\mathbb{Z}, \mathbb{Z}_+, \pm 1)$ and $\mathcal{D}_k$ for the syntactic monoid of the language $\mathcal{D}_k$.

Theorem 57. *Given a finite monoid M , the procedure below correctly determines if M divides an iterated typed wreath product of $(\mathbb{Z}, \mathbb{Z}_+)$.*

Recall the definition of $\mathcal{R}$ classes and the $\prec_{\mathcal{R}}$ order from definition 22. We now state our decision procedure:

Definition 58 (Decision Procedure). *Our input is a finite monoid M . Let $R_1, R_2, \dots, R_\ell$ be the $\mathcal{R}$ -classes of M , such that $R_i \prec_{\mathcal{R}} R_j \Rightarrow i > j$. For $i = 1, 2, \dots, \ell$, we maintain relational morphisms ϕ_i from M to an iterated wreath product of $\mathcal{D}_k$ and $\mathbb{U}_1$. We maintain the invariant that*

$$\{m\} = \phi_i^{-1}(\phi_i(m)), \forall m \in R_1, \dots, R_i \quad (\text{Invariant}(i))$$

Let N_0 be the trivial monoid and $\phi_0 : M \triangleleft N_0$ be the trivial relational morphism. For $i = 1, \dots, \ell$, do

⁴We note that S_k itself is not a monoid, because truncated addition is not associative. The monoid $\mathcal{D}_k$ describes the monoid of bounded incrementing/decrementing operations on S_k . Together, they provide an algebraic model of bounded counting sufficient for our purposes.

1. *Note: Here, we record, for any path through the category going into R_i , which arrow was used for passing into it. We will later refer to this annotation as the “entry point coordinate”. It intuitively tells us via which element of R_i we first entered it. Later, $E_{i,j}$ will have multiple strongly connected components (or bonded components, the term used by Tilson) corresponding to R_i , indexed by the different possible entry points (Lemma 63).*

Let $\delta : \text{Obj}(D_{\phi_{i-1}}) \rightarrow \{0, 1\}$ be given as

$$\delta(o) = \begin{cases} 1 & \exists m \in \phi_{i-1}^{-1}(o) : m \preceq_R R_i \\ 0 & \text{else} \end{cases}$$

Let $\partial_i = \{(o \rightarrow_{s,t} ot) : \delta(o) = 0, \delta(ot) = 1\}$.

Define $\phi' : D_{\phi_{i-1}} \triangleleft B$, where B is the left-zero semigroup with identity $*$ adjoined, with left-zeros indexed by the elements of ∂_i ⁵, as the relational morphism generated by (i.e., the intersection of all relational morphisms satisfying)

$$\phi'(o \rightarrow_{(s,t)} ot) \supseteq \begin{cases} \partial_i & \delta(o) = \delta(ot) = 1 \\ \{*\} & \delta(o) = \delta(ot) = 0 \\ \{(o \rightarrow_{(s,t)} ot)\} & \delta(o) = 0, \delta(ot) = 1 \end{cases} \quad (8)$$

where we note $\delta(o) = 1, \delta(ot) = 0$ cannot occur.

By definition, $* \notin \phi'(1 \rightarrow_{(s,o)} o)$ iff $s \preceq_R R_i$.

We note that B is $\mathcal{R}$ -trivial; hence it divides an iterated wreath product of U_1 .

This is a relational morphism.

Set $\phi_{i,1} := \text{Ext}(\phi', \phi_{i-1}) : M \triangleleft N_{i,1}$ where $N_{i,1} = B \circ N_{i-1}$.

2. Build a sequence of relational morphisms $\phi_{i,2}, \phi_{i,3}, \dots$ as follows, over $j = 1, 2, 3, \dots$:

- (a) *Intuition: We simplify $D_{\phi_{i,j}}$ in two ways. First, all information from earlier annotation is removed once we have entered R_i . This simplifies our proof by avoiding any complications arising from interactions between old and new annotation. Second, elements of $\mathcal{D}_k$ provide extra information beyond bounded counting; here, we project its elements to the underlying set of truncated integers.*

Write an element of $N_{i,j}$ as

$$a = (f_{i,j}, \dots, f_{i,1}, f).$$

where $f_{i,j} : N_{i,j-1} \rightarrow \mathcal{D}_k$. Define its “retained signature” by

$$\text{sig}_{i,j}(a) := \underbrace{(0 \cdot \underbrace{f_{i,j}(1_{N_{i,j-1}})}_{\in \mathcal{D}_k}, \dots, 0 \cdot \underbrace{f_{i,2}(1_{N_{i,1}})}_{\in B}, \underbrace{f_{i,1}(1_{N_{i-1}})}_{\in B})}_{\in S_k},$$

where we view $\mathcal{D}_k$ as acting on $S_k := [-k, \dots, k] \cup \{\perp\}$ from the right (7). We define a right congruence⁶ $\rho_{i,j}$ on $N_{i,j}$ by setting $a \rho_{i,j} b$ whenever (i) $a = b$, or (ii) $f_{i,1}^a(1_{N_{i-1}}) \neq *, f_{i,1}^b(1_{N_{i-1}}) \neq *$, and $\text{sig}_{i,j}(a) = \text{sig}_{i,j}(b)$. We define $E_{i,j}$ as the category obtained from $D_{\phi_{i,j}}$ and the right-congruence $\rho_{i,j}$ (based on the discussion in Section H.2).

- (b) Consider the set $\Omega_{i,j}$ of algebraic relational morphisms $\phi' : E_{i,j} \triangleleft \mathbb{Z}$ such that for each $s \in R_i$ and any $t \in \phi_{i,j}(s)$, $\phi'(1 \rightarrow_{s,t} t)$ is a finite set.⁷ *Note: In the automaton-based*

⁵ $B := \partial_i \cup \{*\}$, $*b = b$, $rb = r$ whenever $r \in \partial_i$, $b \in B$.

⁶In the sense that $apb \Rightarrow arpb$ for any $a, b, r \in N_{i,j}$.

⁷Without loss of generality, we restrict to relational morphisms ϕ' where, for each object o with non- $*$ entry annotation, every value of ϕ' on an arrow $1 \rightarrow o$ is generated by factorization through a prefix ending inside R_i : $\phi'(\alpha) = \bigcup_{\alpha = \beta\gamma, \beta \in \pi^{-1}(R_i)} \phi'(\beta)\phi'(\gamma)$. This condition is imposed because relational morphisms may add values that are not generated by any nontrivial factorization of an arrow. The construction in Lemma 69 satisfies this condition.

proof, the key requirement is for the relabelings to be balanced. Here, we formulate this in terms of finite sets.

- (c) If there is $\phi' \in \Omega_{i,j}$ such that at least one of the following is satisfied:
- i. Condition A: $\phi' : E_{i,j} \triangleleft \mathbb{Z}$ assigns disjoint images to two arrows in $\text{Hom}(o \rightarrow o')$ where o, o' both are in R_i (in the sense of Definition 59).
 - ii. Condition B: There exist objects $x, y \in \text{Obj}(E_{i,j})$, arrows $a \in \text{Hom}(x \rightarrow y); b \in \text{Hom}(y \rightarrow x); \ell \in \text{Hom}(x \rightarrow x)$ such that x, y are inside R_i^8 , but no arrow in $\text{Hom}(x, y)$ ends in R_i^9 , whereas ℓ ends in R_i^{10} and
$$\phi'(ab) \cap \phi'(\ell) = \emptyset.$$
then we take such a ϕ' , turn it to a relational morphism $\tilde{\phi}' : E_{i,j} \triangleleft \mathcal{D}_k^{11}$, pull it back to $\hat{\phi}' : D_{\phi_{i,j}} \triangleleft \mathcal{D}_k$ and build $\phi_{i,j+1} := \text{Ext}(\hat{\phi}', \phi_{i,j}) : M \triangleleft N_{i,j+1}$, where $N_{i,j+1} := \mathcal{D}_k \circ N_{i,j}$.
- (d) Otherwise, we terminate this inner loop.
- (e) One possibility is that $\phi_{i,j}^{-1}(\phi_{i,j}(m)) = \{m\}$ for each $m \in R_i$. By induction, we have satisfied Invariant(i). Then, set $\phi_i := \phi_{i,j} : M \triangleleft N_i$ and pass to the next iteration in the outer loop. Else, we exit from the entire algorithm and declare failure.

We declare success and return N_ℓ if we iterated through all $i = 1, \dots, \ell$ without ever declaring failure.

Note: Motivation of this algorithm: The generic strategy would be to keep generating relational morphisms to $\mathbb{Z}$, but that would not suffice for decidability, because (i) we wouldn't know how often to iterate, and (ii) wouldn't know which types to use. We "guide" the process by (i) proceeding along $\mathcal{R}$ classes, (ii) considering morphisms to well-selected finite monoids, (iii) passing to $E_{i,j}$ instead of the more complicated $D_{\phi_{i,j}}$, and (iv) focusing on the sets $\Omega_{i,j}$.

Throughout, we write $1 \in \text{Obj}(E_{i,j})$ for the object arising from the identity element of $N_{i,j}$.

Definition 59. For any of the categories $E_{i,j}$ constructed in the decision procedure, we define the following notions. Let $\pi : E_{i,j} \prec M$ be the canonical division.

1. Let S_i be the set of arrows $o \rightarrow_{s,t} o'$ such that there are arrows $1 \rightarrow_{s',o} o$ and $o' \rightarrow_{s'',o'} o'tt''$ such that $ss's'' \in R_i$.
2. We say an arrow $\alpha \in \text{Hom}(o \rightarrow o')$ "ends" in R_i if there is an arrow $\beta \in \text{Hom}(1 \rightarrow o)$ such that $\beta\alpha$ is defined and $\beta\alpha \in \pi^{-1}(R_i)$.
3. We say an arrow $\alpha \in \text{Hom}(o \rightarrow o')$ "starts" in R_i if there is an arrow $\beta \in \text{Hom}(1 \rightarrow o)$ such that $\beta\alpha$ is defined and $\beta \in \pi^{-1}(R_i)$.
4. We say an object o is "inside R_i " if $\text{Hom}(1 \rightarrow o) \cap \pi^{-1}(R_i) \neq \emptyset$.

H.3.1 Key Properties of the Procedure

Lemma 60 (Correctness). If the algorithm succeeds, then M divides an iterated wreath product of $(\mathbb{Z}, \mathbb{Z}_+)$, of the form $((\dots \mathbb{Z}) \circ \mathbb{Z}) \circ \mathbb{Z}$.

⁸In the sense of Definition 59: $\text{Hom}(1, x) \cap \pi^{-1}(R_i) \neq \emptyset; \text{Hom}(1, y) \cap \pi^{-1}(R_i) \neq \emptyset$, where $\pi : E_{i,j} \triangleleft M$ is the canonical division.

⁹In the sense of Definition 59: $\forall \beta \in \text{Hom}(x \rightarrow y); \text{Hom}(1 \rightarrow x) \beta \cap \pi^{-1}(R_i) = \emptyset$, where $\pi : E_{i,j} \triangleleft M$ is the canonical division.

¹⁰In the sense of Definition 59: $\text{Hom}(1, x) \ell \cap \pi^{-1}(R_i) \neq \emptyset$,

¹¹For sufficiently large k , greater than the absolute value of, for each $s \in R_i$ and any $t \in \phi_{i,j}(s)$, the entries of the finite set $\phi'(1 \rightarrow_{s,t} t)$. To convert a relational morphism to $\mathbb{Z}$ into one on $\mathcal{D}_k$, map 1 to a , -1 to b , etc., and close to make it a relational morphism. Choosing k large enough will ensure $\perp$ is avoided on these homsets. Also, on these homsets, this resulting relational morphism is at least as finegrained as the original one, since one can get back the original one by mapping Dyck elements to their height.

Proof. We refer to a wreath product of the form $(\dots \circ \cdot) \circ \cdot) \circ \cdot) \circ \cdot$ as *left-associative*. First, if the algorithm succeeds we obtain a relational morphism $\phi_k : M \triangleleft N$ where N divides a left-associative wreath product of U_1 and $\mathcal{D}_k$. By the invariant maintained in the Decision Procedure, ϕ_k is injective, making it a division. We now need to explain why M also divides a product of the form $((\dots \mathbb{Z}) \circ \mathbb{Z}) \circ \mathbb{Z}$. First, by we can obtain a representation in terms of $\mathcal{D}_k \preceq ((U_1)^{2k} \circ \mathbb{Z})$ and U_1 , in the left-associative bracketing due to the associativity of the finite wreath product at the level of pseudovarieties (Tilson, 1987). By repeated application of lemma 61, we then obtain a left-associative wreath product where the factors are either direct products of U_1 or just $\mathbb{Z}$. Because direct products divide wreath products, we know that $M \preceq (S \circ (N \times U_1)) \circ T$ implies $M \preceq ((S \circ N) \circ U_1) \circ T$. Iteratively applying this identity, we obtain a division into a left-associative wreath product of U_1 and $\mathbb{Z}$ (with much greater depth). Finally because $U_1 \preceq \mathbb{Z}$, we can conclude that M divides a left-associative wreath product of $\mathbb{Z}$. □

Lemma 61. For typed monoids T, S , we have that $T \circ (U_1 \circ S) \preceq (T \circ U_1^{|\mathfrak{T}_S|}) \circ S$

Proof. We show the proof ignoring any constants in the wreath product to put less strain on notation, but it is easy to add them back in. Let $(f_1, (g_1, h_1))(f_2, (g_2, h_2)) \cdots (f_n, (g_n, h_n))$ be a multiplication in $T \circ (U_1 \circ S)$ where $h_i \in S$, $g_i \in U_1^S$, and $f_i \in T^{U_1^S \times S}$. We will write multiplication in all monoids using Π , but assume the distinction is clear. By lemma 37 we can compute the type of this multiplication by computing the following and checking the types of the indicated terms 1, 2, and 3

$$\left(\underbrace{\prod_{i=1}^{n-1} f_i \left(\left(\prod_{j=1}^{i-1} g_j \left(\prod_{k=1}^{j-1} h_k \right), \prod_{j=1}^{i-1} h_j \right) \right)}_1, \underbrace{\left(\prod_{i=1}^{n-1} g_i \left(\prod_{j=1}^{i-1} h_j \right), \prod_{i=1}^{n-1} h_i \right)}_{\substack{2 \\ 3}} \right)$$

We will simulate this computation in $(T \circ U_1^{|\mathfrak{T}_S|}) \circ S$. That is, the desired computation is given by the following injective relational morphism given below. Intuitively, $U_1^{|\mathfrak{T}_S|}$ records which of the finitely many *types* have been seen. Since f must be type-respecting, this suffices to recreate the computation of $T \circ (U_1 \circ S)$.

$$\begin{aligned} \phi: T \circ (U_1 \circ S) &\rightarrow (T \circ U_1^{|\mathfrak{T}_S|}) \circ S \\ \alpha_{f,g} &\in (T^{U_1^{|\mathfrak{T}_S|}} \times U_1^{|\mathfrak{T}_S|})^S \\ \gamma_{f,\zeta} &\in T^{U_1^{|\mathfrak{T}_S|}} \\ \phi((f, (g, h))) &= (\alpha_{f,g}, h) \\ \alpha_{f,g}(\zeta) &= (\gamma_{f,\zeta}, (\lambda_{\mathfrak{S}_1}, \lambda_{\mathfrak{S}_2}, \dots, \lambda_{\mathfrak{S}_{|\mathfrak{T}_S|}})) & \lambda_{\mathfrak{S}} &= \begin{cases} 1 & \zeta \in \mathfrak{S} \\ 0 & \zeta \notin \mathfrak{S} \end{cases} \\ \gamma_{f,\zeta}((\lambda_{\mathfrak{S}_1}, \lambda_{\mathfrak{S}_2}, \dots, \lambda_{\mathfrak{S}_{|\mathfrak{T}_S|}})) &= f(\beta, \zeta) & \text{Im}(\beta) &\subseteq \bigcap_{\mathfrak{S} \in \mathfrak{T}_S; \lambda_{\mathfrak{S}}=1} \mathfrak{S} \end{aligned}$$

Note that $\gamma_{f,\zeta}$ is well-defined because $f((\beta, \zeta))$ is identical for all β whose images are contained in the same set of types, since f is type-respecting. Now, this mapping is injective because every type-respecting function $g \in U_1^S$ is represented by some $(\lambda_{\mathfrak{S}_1}, \lambda_{\mathfrak{S}_2}, \dots, \lambda_{\mathfrak{S}_{|\mathfrak{T}_S|}}) \in U_1^{|\mathfrak{T}_S|}$, and every type-respecting function f is represented as some $\gamma_{f,\zeta}$.

Thus the sequence $(f_1, (g_1, h_1))(f_2, (g_2, h_2)) \cdots (f_n, (g_n, h_n))$ maps homomorphically to the sequence $(\alpha_{f_1, g_1}, h_1)(\alpha_{f_2, g_2}, h_2) \cdots (\alpha_{f_n, g_n}, h_n)$ which is evaluated as

$$\left(\prod_{i=1}^{n-1} \alpha_{f_i, g_i} \left(\prod_{j=1}^{i-1} h_j \right), \underbrace{\prod_{i=1}^{n-1} h_i}_3 \right)$$

Observe that the type 3 is already computed in the second coordinate. We will see that 1 and 2 are also computed in the first coordinate:

$$\begin{aligned} \prod_{i=1}^{n-1} \alpha_{f_i, g_i} \left(\prod_{j=1}^{i-1} h_j \right) &= \alpha_{f_1, g_1} (0) \alpha_{f_2, g_2} (h_1) \cdots \alpha_{f_n, g_n} \left(\prod_{j=1}^{n-1} h_j \right) \\ &= (\gamma_{f_1, 0}, \Lambda_1) (\gamma_{f_2, h_1}, \Lambda_2) \cdots (\gamma_{f_n, \prod_{i=1}^{n-1} h_i}, \Lambda_n) \\ &= \left(\underbrace{\left(\prod_{i=1}^{n-1} f_i \left(1, \prod_{j=1}^{i-1} h_j \right) \right)}_1, \underbrace{\prod_{i=1}^{n-1} g_i \left(\prod_{j=1}^{i-1} h_j \right)}_2 \right) \end{aligned}$$

□

The following properties of $\omega \in \Omega_{i,j}$ are key:

Lemma 62 (Balance on Loops). *Let $\omega \in \Omega_{i,j}$.*

1. *Let $u \in R_i$, $\rho \in M$ such that $u\rho = u$. Consider arrows*

$$1 \rightarrow_{(u, \rho)} o \rightarrow_{(\rho, t)} ot$$

in $E_{i,j}$. Then in fact $ot = o$, and $\omega(o \rightarrow_{(\rho, t)} o) = \{0_{\mathbb{Z}}\}$ where $0_{\mathbb{Z}}$ is the neutral element of $\mathbb{Z}$.

2. *Let u be such that $R_i u = R_i$.*

For $v \in R_i$, consider arrows

$$1 \rightarrow_{(v, \rho)} o \rightarrow_{(u, t)} ot$$

in $E_{i,j}$. Then $\omega(o \rightarrow_{(u, t)} ot)$ is a singleton.

Proof. For the first point, we have two claims to prove: that (a) $ot = o$, and that (b) the image is $\{0_{\mathbb{Z}}\}$. We first note that (a) holds at $j = 1$: $u \in R_i$, $u\rho = u$ and $1 \rightarrow_{(u, \rho)} o \rightarrow_{(\rho, t)} ot$ in $E_{i,j}$, then in fact $ot = o$ by construction of $E_{i,1}$. We also note that, if (b) has been shown for $1, \dots, j$, then (a) follows for $1, \dots, j+1$. We thus need to perform the inductive step for (b). We consider the arrow $o \rightarrow_{(\rho, t)} o$. Assume $z \in \omega(o \rightarrow_{(\rho, t)} o)$ and $z \neq 0$. Now, for all $k \geq 0$, $u\rho^k = u$; hence, $\omega(1 \rightarrow_{(u, \rho)} o)$ is infinite. This is a contradiction to $\omega \in \Omega_{i,j}$.

For the second point, from $R_i u = R_i$, obtain ρ such that $v\rho = v$. We consider arrows:

$$1 \rightarrow_{(v, \rho)} o \rightarrow_{(u, t)} ot \rightarrow_{(\rho, t')} ott'$$

By (1), in fact, $ott' = o$ and $\omega(o \rightarrow_{(u, t)} ot \rightarrow_{(\rho, t')} ott') = \{0_{\mathbb{Z}}\}$. Because ω is a relational morphism and $\mathbb{Z}$ is a group, this entails $\omega(o \rightarrow_{(u, t)} ot)$ must be a singleton. □

We deduce the following structural properties of $E_{i,j}$, which are used both for termination and for completeness of the procedure. As foreshadowed when we defined the “entry-point coordinate” $\phi_{i,1}$, we find that R_i is reflected in $E_{i,j}$ in multiple strongly connected components (or “bonded components”), each indexed by a different entry arrow of the “boundary set” ∂_i we defined there:

Lemma 63 (Structure of $E_{i,j}$).

1. Let $\alpha := (o \rightarrow_{(s,t)} ot) \in \partial_i$, and let $m \in R_i$. Then there is exactly one $u \in \text{Obj}(E_{i,j})$ such that both of the following hold: (i) there is an arrow $1 \rightarrow_{(m,u)} u$, and (ii) the $\phi_{i,1}$ component of u is α .
2. Let $\alpha = (1 \rightarrow_{(m,u)} u), \alpha' = (1 \rightarrow_{(m',u')} u')$ be arrows in $E_{i,j}$, where $m, m' \in R_i$; and assume u, u' have the same $\phi_{i,1}$ component. Then there is $\beta \in \text{Hom}(u, u')$ such that $\alpha' = \alpha\beta$.

Proof. We show this by induction over j . The claims are immediate at $j = 1$. For the first claim, we show the inductive step using Lemma 62. For any arrow

$$\alpha = (1 \rightarrow_{(m,o)} o)$$

with $m \in R_i$, Lemma 62.2 entails that ω is single-valued on all arrows $\beta = (o \rightarrow_{(m',t)} ot)$ with $mm' \in R_i$; hence, inductively, after passing to $E_{i,j+1}$, only a single arrow is derived from β .

The second claim follows by choosing n such that $m' = mn$; the first claim enforces the presence of an arrow covering n in $\text{Hom}(u, u')$. $\square$

Lemma 64 (Termination). *The inner loop always terminates.*

Proof. A new morphism $\phi_{i,j}$ can only be accepted because it separates two arrows in a homset $\text{Hom}(o \rightarrow o')$ where o, o' are both inside R_i . We need to show that this can only happen a bounded number of times.

To this end, we first note by the first point of Structure of $E_{i,j}$ that, for each entry arrow α , we can uniquely assign a single o for every $r \in R_i$, and this mapping is a surjection onto the objects inside R_i .

Also, with every j iteration, this structure becomes finer, i.e., a homset $\text{Hom}(o \rightarrow o')$ where o, o' are in R_i can split into several ones that (save for arrows leaving R_i , where arrows carrying the same M -label might go into into different descendant homsets) each exactly carry some subset of the original arrows. Thus, the total number of times this happens must be bounded. $\square$

Lemma 65 (Computability). *In each step of the inner loop, we can effectively decide if there is a ϕ' satisfying the requirements.*

Proof. We can always construct a relational morphism of the second type if it exists. We can always apply this case when it is available. We need to check when a relational morphism of the first type exists. For the arrows that start and end in R_i , since they will be single-valued, we can check the $\mathbb{Z}$ -module of functions defined on these arrows satisfying the balancedness condition. For arrows in the connected components of objects in R_i that however do not end R_i , all of them will receive a nonempty set of values by closing this under composition, for, otherwise, we would have been able to choose a relational morphism of the second type. We can extend any solution from a single connected component to a full morphism, by the Extension Lemma (Lemma 69). $\square$

H.3.2 Establishing Completeness

Assume that, after constructing $E_{i,j}$, the procedure terminates with failure. We localize this failure into a smaller, focused category.

Definition 66. For each object o inside R_i , we consider the strongly connected component (or “bonded component” in *Tilson (1987)*) of o :

$$\text{Obj}(C_o) := \{o' \in \text{Obj}(E_{i,j}) : \text{Hom}(o \rightarrow o') \neq \emptyset, \text{Hom}(o' \rightarrow o) \neq \emptyset\}$$

As above, let π be the canonical division $\pi: E_{i,j} \prec M$. Within any homset $\text{Hom}(o' \rightarrow o'')$ in C_o , we merge any two arrows β, β' such that, in $E_{i,j}$, $\text{Hom}(1 \rightarrow o')\beta$ and $\text{Hom}(1 \rightarrow o')\beta'$ both are not in $\pi^{-1}(\bigcup_{j \leq i} R_j)$. By construction, C_o is a well-defined category, and $C_o \prec E_{i,j} \prec M$.

Lemma 67. Every homset $\text{Hom}(u \rightarrow u')$ in C_o hosts an arrow ending in R_i .

Proof. Otherwise, a morphism ϕ' satisfying the second condition would have been selected. $\square$

Lemma 68. Failure of the algorithm entails that some C_o contains a non-singleton homset, i.e., C_o is not trivial.

Proof. Failure of the algorithm implies that there is $m \in R_i$ such that $\phi_{i,j}^{-1}(\phi_{i,j}(m)) \supsetneq \{m\}$.

Now consider $m' \in M, m \neq m'$ such that there is some $\delta \in \phi_{i,j}(m) \cap \phi_{i,j}(m')$.

Let o be the corresponding element of $E_{i,j}$.

This means that, in $E_{i,j}$, there are distinct arrows $1 \rightarrow_{(m,\delta)} o$ and $1 \rightarrow_{(m',\delta)} o$.

Now we consider the $\phi_{i,1}$ coordinate of o ("entry point coordinate"); this is of the form $(u \rightarrow_{(s,t)} v)$ corresponding to some arrow of $D_{\phi_{i-1}}$.

By the inductive hypothesis "(Invariant(i))", $\text{Hom}(1 \rightarrow u)$ in that category had exactly one object $1 \rightarrow_{(s',u)} u$, where $s' \in R_k$ for some $k < i$.

We can thus factorize

$$1 \rightarrow_{(m,\delta)} o = \left(1 \rightarrow_{(s',u)} u\right) \left(u \rightarrow_{(s,t)} v\right) \left(v \rightarrow_{(w,\dots)} o\right) \quad (9)$$

$$1 \rightarrow_{(m',\delta)} o = \left(1 \rightarrow_{(s',u)} u\right) \left(u \rightarrow_{(s,t)} v\right) \left(v \rightarrow_{(w',\dots)} o\right) \quad (10)$$

Now let v be the corresponding entry object, which must be contained in C_o (by Lemma 63.1); then (by Lemma 63.2) there is an arrow $\alpha = 1 \rightarrow_{(s',v)} v$ such that there are $\beta, \beta' \in \text{Hom}(v, o)$ with $1 \rightarrow_{(m,\delta)} o = \alpha\beta$ and $1 \rightarrow_{(m',\delta)} o = \alpha\beta'$. We thus have shown $\text{Hom}(v, o)$ to be nonsingleton. $\square$

Lemma 69 (Extension Lemma). Assume $\omega: C_o \triangleleft \mathbb{Z}$ stays finite-image on all arrows in C_o that end in R_i . Then there is an algebraic relational morphism $\tilde{\omega}: E_{i,j} \triangleleft \mathbb{Z}$ such that $\tilde{\omega}(1 \rightarrow_{(s,t)} t)$ is a singleton set whenever $s \in R_i$, and $\tilde{\omega}|_{C_o} \equiv \omega$.

Proof. There are two aspects here: achieving $\tilde{\omega}|_{C_o} \equiv \omega$ – this is easy because C_o is a bonded component – and achieving that $\tilde{\omega}(1 \rightarrow_{(s,t)} s)$ is a finite set for $s \in R_i$ – which requires care.

We first define

$$\omega_*(\alpha) := \begin{cases} \omega(\alpha) & \alpha \in C_o \\ \{0\} & \text{else} \end{cases} \quad (11)$$

which is not in general a relational morphism. We define $\tilde{\omega}$ as the closure under composition, so that $\tilde{\omega}$ is a relational morphism. Constructively, we can write

$$\tilde{\omega}(\alpha) = \bigcup_{\gamma_1 \dots \gamma_n = \alpha} \sum_{i=1}^n \omega_*(\gamma_i) \quad (12)$$

where " $\sum$ " is to be understood in a set-valued sense as $A + B = \{a + b : a \in A, b \in B\}$.

Because C_o is a strongly connected component, $\tilde{\omega}|_{C_o} \equiv \omega$. Now consider $\alpha: 1 \rightarrow_{(s,t)} t$ for some $s \in R_i$. One option is that $t \notin C_o$; in this case, no path through C_o can multiply out to this arrow, and $\tilde{\omega}(\alpha) = \{0\}$, which is a finite set. The other option is that $t \in C_o$.

Assume $\omega(\alpha)$ is infinite. Then, for each n , there is a path of the form:

$$\alpha = \gamma_1 \gamma_2 \gamma_3 \dots \gamma_n \quad (13)$$

where $\sum_{i=2}^n \omega_*(\gamma_i)$ contains unboundedly (positive or negative) large numbers; we can WLOG choose this so that $\omega_*(\gamma_2) \neq \{0\}$ by multiplying the initial prefix out into γ_1 if needed. By construction, $\gamma_2 \in C_o$; hence, γ_1 must end in C_o . But then we can set $\beta := \gamma_2 \dots \gamma_n$, which starts and ends in C_o , and also ends in R_i because α does. But then $\tilde{\omega}(\beta(n))$ has unboundedly large numbers as $n \rightarrow \infty$, which is a contradiction to $\tilde{\omega}|_{C_o} \equiv \omega$. $\square$

Lemma 70. Consider two arrows in the same homset in C_o : $\beta_1 := u \rightarrow_{(s,t)} ut$ and $\beta_2 := u \rightarrow_{(s',t')} ut$. If $\beta_1 \neq \beta_2$, then

1. there is $\lambda \in R_i \cap \pi(\text{Hom}(1 \rightarrow u))$ such that $\lambda s \neq \lambda s'$.
2. there is $\alpha \in \text{Hom}(o \rightarrow u)$ such that $\alpha \beta_1 \neq \alpha \beta_2$.

Proof. This follows from the derived category construction; the right-congruences applied afterwards preserve it. $\square$

Lemma 71 (Completeness). If the procedure terminates with failure, M does not divide an iterated wreath product of $(\mathbb{Z}, \mathbb{Z}_+)$.

Proof. Assume that, after constructing $E_{i,j}$, the procedure terminates with failure.

Setting up a division Choose as C'' one nontrivial C_o (it doesn't matter which one); from now on this fixes o . For the purposes of defining typed division, we view C_o as rooted in o (Definition 75). Assume M divides an iterated wreath product of $(\mathbb{Z}, \mathbb{Z}_+)$, then so does C'' (in the sense of Definition 75) by Lemma 77. Consider a division of a minimum-depth wreath product of $\mathbb{Z}$, $\psi : C'' \prec \underbrace{\mathbb{Z} \circ \dots \circ \mathbb{Z}}_{T \text{ times}}$ (here all $\mathbb{Z}$ are typed like $(\mathbb{Z}, \mathbb{Z}_+)$); in particular,

because C'' is not trivial, $T \geq 1$. Let $\omega = \pi^{(T)} \circ \psi$; by definition, $\omega : C'' \triangleleft \mathbb{Z}$. Our goal is to create a division $\mu : C'' \prec \underbrace{\mathbb{Z} \circ \dots \circ \mathbb{Z}}_{T-1 \text{ times}}$ (again in the sense of Definition 75). This would

be a contradiction to the minimality of T . As a consequence, C'' cannot divide *any* wreath product of $(\mathbb{Z}, \mathbb{Z}_+)$.

Note: The basic idea is to show that ω cannot have provided any useful information, and in particular gives the same values across different arrows in a given hom-set. Either it stays finite-image on all arrows going into R_i (in which case ω provides as much information as an element of $\Omega_{i,j}$ – and the algorithm having terminated indicates that no such element would have been helpful), or it is infinite-image (in which case we can pass to a limiting element $\pm\infty$, showing that ω contributes no information useful to $(\mathbb{Z}, \mathbb{Z}_+)$ -based recognizers).

Case 1: ω stays finite-image on all arrows ending in R_i inside C'' First, assume ω stays finite-image on all arrows ending in R_i inside C'' . We obtain $\tilde{\omega} : E_{i,j} \triangleleft \mathbb{Z}$ via Lemma 69. We observe that $\tilde{\omega} \in \Omega_{i,j}$; hence, $\tilde{\omega}$ must assign singleton images to any arrow in C_o that ends inside R_i . The algorithm having terminated means that $\tilde{\omega}$, and hence ω , cannot assign distinct images to any two arrows staying inside R_i that appear in a single homset in C_o ; also, any of these needs to be mapped to a singleton set. It also means that, when considering an arrow that is in C_o but ends outside of R_i , it might in fact be assigned some further set of values, but it must overlap with the value assigned to the arrows staying inside R_i . That is, in any homset, there is one single value shared across all arrows; also, there potentially is a further set of values assigned to the arrows that end outside of R_i .

For each homset $\text{Hom}(u \rightarrow u')$, we obtain a unique $\theta_{u,u'} \in \mathbb{Z}$ such that $\omega(\alpha) = \{\theta_{u,u'}\}$ whenever $\alpha \in \text{Hom}(u \rightarrow u')$ ends in R_i ; also, when the homset contains an arrow leaving

R_i (we can write it as $\beta : u \rightarrow_{(\perp, \dots)} u'$, $\theta_{u,u'} \in \omega(\beta)$). We write $v_u := \theta_{o,u}$; importantly, $v_{ut} = v_u \theta_{u,ut}$.

We obtain the desired division $\mu : C_o \prec \underbrace{(\mathbb{Z} \circ \dots \circ \mathbb{Z})}_{T-1 \text{ times}}$ as follows. Note that any object in the image of ψ has the form $(g, \dots)$ where $g : \mathbb{Z} \rightarrow \underbrace{(\mathbb{Z} \circ \dots \circ \mathbb{Z})}_{T-1 \text{ times}}$. For any arrow $u \rightarrow_{(s,t)} ut$ in C_o , we define:

$$\mu(u \rightarrow_{(s,t)} ut) := \{g(v_u) : (g, \theta_{u,ut}) \in \psi(u \rightarrow_{(s,t)} ut)\} \quad (14)$$

which is a nonempty subset of $\underbrace{(\mathbb{Z} \circ \dots \circ \mathbb{Z})}_{T-1 \text{ times}}$.

We first show that μ is an algebraic relational morphism. Consider arrows $\alpha : u \rightarrow_{(s,t)} ut$ and $\beta : ut \rightarrow_{(s',t')} utt'$ in C_o . Then:

$$\begin{aligned} \mu(\alpha)\mu(\beta) &= \{g(v_u) : (g, \theta_{u,ut}) \in \psi(\alpha)\} \{g'(v_{ut}) : (g', \theta_{ut,utt'}) \in \psi(\beta)\} \\ &= \{g(v_u)g'(v_{ut}) : (g, \theta_{u,ut}) \in \psi(\alpha), (g', \theta_{ut,utt'}) \in \psi(\beta)\} \\ &\subseteq \{g(v_u) : (g, \theta_{u,utt'}) \in \psi(\alpha\beta)\} \\ &= \mu(\alpha\beta) \end{aligned}$$

where the “ $\subseteq$ ” step used the fact that ψ is an algebraic relational morphism.

We, second, show that μ is a typed division. Consider two arrows $\beta_1 := u \rightarrow_{s,t} ut$, $\beta_2 := u \rightarrow_{s',t} ut$ in C'' ; if they are distinct, we find $\alpha \in \text{Hom}(o \rightarrow u)$ such that $\alpha\beta_1 \neq \alpha\beta_2$ (Lemma 70). Then there are disjoint types T_1, T_2 of $\underbrace{\mathbb{Z} \circ \dots \circ \mathbb{Z}}_{T \text{ times}}$ such that $\psi(\alpha\beta_1) \subseteq T_1$,

$\psi(\alpha\beta_2) \subseteq T_2$. That is, for any $g_s, g_{s'}$ selected for the two arrows, we have

$$(g_\alpha(\cdot)g_s(\cdot\theta_{o,u}), \theta_{o,u}\theta_{o,ut}) \in \psi(\alpha\beta_1) \subseteq T_1 \quad (15)$$

$$(g_\alpha(\cdot)g_{s'}(\cdot\theta_{o,u}), \theta_{o,u}\theta_{o,ut}) \in \psi(\alpha\beta_2) \subseteq T_2 \quad (16)$$

Hence, there must be disjoint types V_1, V_2 of $\underbrace{\mathbb{Z} \circ \dots \circ \mathbb{Z}}_{T-1 \text{ times}}$ such that

$$\mu(\alpha\beta_1) = g_\alpha(0)g_s(\theta_{o,u}) \in V_1 \quad (17)$$

$$\mu(\alpha\beta_2) = g_\alpha(0)g_{s'}(\theta_{o,u}) \in V_2 \quad (18)$$

But then

$$g_s(\theta_{o,u}) \in g_\alpha(0)^{-1}V_1 \quad (19)$$

$$g_{s'}(\theta_{o,u}) \in g_\alpha(0)^{-1}V_2 \quad (20)$$

where $V_1 \cap V_2 = \emptyset$, the types $g_\alpha(0)^{-1}V_1$ and $g_\alpha(0)^{-1}V_2$ are also disjoint types.¹² The above assumes that T_1, T_2 are defined simply by first-coordinate evaluation at zero; if they instead arise as Boolean combinations of multiple observations, μ would instead map into a direct product of multiple copies where we evaluate g at different coordinates; in this case, the proof here would be lifted to depth- T wreath products of direct products of $\mathbb{Z}$.

Case 2: ω is infinite-image on some arrow ending in R_i inside C'' Let $\alpha = (u_0 \rightarrow_{(s_0,t_0)} u_0 t_0)$ in C'' be an arrow ending in R_i which is associated with an infinite number of different values under ω . Without loss of generality, we may assume that $\sup \omega(\alpha) = +\infty$ (else, $\inf \omega(\alpha) = -\infty$ and we replace $+\infty$ by $-\infty$ below).

¹²Here, we take a more general definition of types that are closed under (left) quotients, in line with the topological perspective on typed monoids (Gehrke & Krebs, 2017). This does not change the underlying expressivity of $\text{wpc}(\mathbb{Z})$.

Note that any object in the image of ψ has the form $(g, \dots)$ where $g : \mathbb{Z} \rightarrow \underbrace{(\mathbb{Z} \circ \dots \circ \mathbb{Z})}_{T-1 \text{ times}}$.

Because g arises from a composition of type-respecting functions, $g(\infty) := \lim_{x \rightarrow \infty} g(x)$ is well-defined and in $\mathbb{Z}$. We define

$$\mu(u \rightarrow_{(s,t)} ut) := \{g(\infty) : (g, \dots) \in \psi(u \rightarrow_{(s,t)} ut)\} \quad (21)$$

which is a subset of $\underbrace{(\mathbb{Z} \circ \dots \circ \mathbb{Z})}_{T-1 \text{ times}}$.

This is an algebraic relational morphism $\mu : C'' \triangleleft \underbrace{(\mathbb{Z} \circ \dots \circ \mathbb{Z})}_{T-1 \text{ times}}$: For arrows β, β' such that $\beta\beta'$ is defined, we have:

$$\begin{aligned} \mu(\beta)\mu(\beta') &= \{g(\infty)h(\infty) : (g, \dots) \in \psi(\beta), (h, \dots) \in \psi(\beta')\} \\ &= \{\lim_{x \rightarrow \infty} (g(x)h(x)) : g, \dots, h, \dots\} \\ &= \{\lim_{x \rightarrow \infty} (g(x)h(x\tau_1)) : g, \dots, h, \dots\} \\ &\subseteq \{G(\infty) : (G, \dots) \in \psi(\beta\beta')\} \\ &= \mu(\beta\beta') \end{aligned}$$

We need to show that it is typed and injective; the proof is similar to Case 1. Consider two distinct arrows in the same homset in C'' :

$$\beta_1 := u \rightarrow_{(s,t)} ut \quad \beta_2 := u \rightarrow_{(s',t')} ut$$

If $\beta_1 \neq \beta_2$, that means there is $\lambda \in R_i \cap \pi^{-1}(\text{Hom}(1 \rightarrow u))$ such that $\lambda s \neq \lambda s'$, by Lemma 70.

We now construct $\rho_1, \rho_2 \in M$ such that

$$\alpha := 1 \rightarrow_{(\rho_1, u_0)} u_0 \rightarrow_{(s_0, t_0)} u_0 t_0 \rightarrow_{(\rho_2, t')} u = 1 \rightarrow_{(\rho_1 s_0 \rho_2, u)} u \quad (22)$$

in $\text{Hom}(1 \rightarrow u)$ satisfies $\lambda \in \pi^{-1}(\alpha)$.¹³ Hence, $\alpha\beta \neq \alpha\beta'$ in $E_{i,j}$. In C'' , $\alpha' := u_0 \rightarrow_{(s_0, t_0)} u_0 t_0 \rightarrow_{(\rho_2, t')} u$ satisfies $\alpha'\beta \neq \alpha'\beta'$. Now, we note that $\sup \omega(\alpha') = +\infty$; this enforces that $\mu(\beta) \cap \mu(\beta') = \emptyset$.

Overall, we have obtained a division μ in either case, and, by contradiction, established that M does not divide an iterated wreath product of $(\mathbb{Z}, \mathbb{Z}_+)$. $\square$

H.3.3 Deriving Main Results

Theorem 15. *Membership of $M \in \text{C-RASP} \cap \mathbf{REG}$ is decidable in $O(\text{poly}(|M|))$ time.*

Proof. We need to show that (i) the algorithm from Definition 58 correctly determines membership in C-RASP (shown in Theorem 57), and (ii) that it runs in polynomial time in $|M|$. For (ii), one route is via the automaton-based formulation of the algorithm in Appendix J, with proof of time polynomial in the number of automaton states (hence the size of the syntactic monoid) in Corollary 88. Another route is by noting that one can identify candidate morphisms $\phi' : E_{i,j} \triangleleft \mathbb{Z}$ by restricting to the strongly-connected component C_o , encoding (i) respecting composition, (ii) zeros on idempotent arrows as linear constraints. The number of such linear constraints is polynomial in the size of M ; an integral basis of the solution space can then be found in polynomial time. A solution can be extended to an element of $\Omega_{i,j}$ via the Extension Lemma 69. $\square$

Corollary 72. $\text{C-RASP} \cap \mathbf{REG} = \text{wpc}(\mathbf{Dy})$.

¹³Choose $\rho_1 \in R_i$ just to produce the first arrow. Now we choose $\rho_2 \in M$ such that $\rho_1 s_0 \rho_2 = \lambda$, using that $\rho_1 s_0 \rho_2$ and λ both are in R_i . Now applying Lemma 62 (Claim 2) inductively to each j , there is an arrow $u_0 t_0 \rightarrow_{(\rho_2, t')} u$.

Proof. The “ $\supseteq$ ” direction is implied by the proof of Lemma 60. The “ $\subseteq$ ” direction follows because the algorithm from Definition 58, when it succeeds on a monoid M , supplies a division $M \prec \mathcal{D}_{k_1} \circ \dots \circ \mathcal{D}_{k_r}$. $\square$

H.4 Defining Division between Categories and Typed Monoids

Here, we define relational morphisms and divisions between finite categories and typed monoids. The definitions are naturally typed extensions of the usual definitions for finite categories (Tilson, 1987).

Definition 73. An algebraic relational morphism $\phi : X \triangleleft M$ from a category X to a monoid M specifies, for each arrow α in X a nonempty set $\phi(\alpha) \subseteq M$, such that $\phi(\alpha)\phi(\beta) \subseteq \phi(\alpha\beta)$ when $\alpha \in \text{Hom}(o \rightarrow o')$, $\beta \in \text{Hom}(o' \rightarrow o'')$, and $1_M \in \phi(1_o)$ where $1_o \in \text{Hom}(o \rightarrow o)$ is the local identity arrow.

This definition matches the definition of relational morphisms in Tilson (1987) in the special case where the target is a monoid; we add the qualifier “algebraic” to highlight that type structure does not yet enter its definition. We do the same for division:

Definition 74. An algebraic division $\psi : X \preceq N$ is an algebraic relational morphism $\psi : X \triangleleft N$ where, for any two distinct arrows $\alpha, \alpha' \in \text{Hom}(o \rightarrow o')$, we have $\psi(\alpha) \cap \psi(\alpha') = \emptyset$.

We expand this definition of division to the case where type structure is present:

Definition 75. Let C be a finite category and let $S = (S, \mathfrak{T}_S, \mathcal{E}_S)$ be a typed monoid. A typed division $\psi : C \preceq S$ is an algebraic division $\psi : C \triangleleft S$, such that, for each homset $H = \text{Hom}_C(o \rightarrow o')$, for each $\alpha \in H$, there is a type $T \in \mathfrak{T}_S$ such that $\psi(\alpha) = T \cap \psi(H)$.

We first note that, in the case of finite monoids with discrete types (i.e., each subset of the monoid is a type), the notions of typed relational morphisms and typed divisions reduce to the usual definitions from Tilson (1987). We can further link the definition to typed recognition,¹⁴ considering the setting of a finite monoid M such that any language it recognizes (in the ordinary sense) via the surjective morphism η is also recognized (in the typed sense) by S . Then:

Lemma 76. Let M be a finite monoid, and let S be a typed monoid. Let $\eta : \Sigma^* \twoheadrightarrow M$ be a surjective morphism, and $h : \Sigma^* \rightarrow S$ a morphism. Assume that for every $P \subseteq M$, there is $T_P \in \mathfrak{T}_S$ such that

$$\eta^{-1}(P) = h^{-1}(T_P).$$

Then $M \preceq S$ in the sense of Definition 75, viewing M as a single-object category.

Proof. Consider the relation $\psi \subseteq M \times S$ defined by $\psi := h \circ \eta^{-1}$. Because η is surjective, this is an algebraic relational morphism. The assumption also ensures that it is an algebraic division. Then, for any $\alpha \in M$, there is a type $T_\alpha \in \mathfrak{T}_S$ such that $\eta^{-1}(\alpha) = h^{-1}(T_\alpha)$. This entails $\psi(\alpha) = h(\eta^{-1}(\alpha)) = h(h^{-1}(T_\alpha)) = T_\alpha \cap \psi(M)$. $\square$

Thus, our definitions here are compatible with the relevant pre-existing definitions from prior work.

We now verify compatibility of our extended notion of division with composition.

Lemma 77. Let C be a finite category, let M be a finite monoid, and let $S = (S, \mathfrak{T}_S, \mathcal{E}_S)$ be a typed monoid. Assume that M and S satisfy the assumptions of Lemma 76. Assume $C \preceq M$ (in the sense of Tilson (1987)). Then $C \preceq S$ (in the sense of Definition 75).

We will apply this to the setting where M is the syntactic monoid of a regular language that is definable in C-RASP; this then provides an iterated wreath product of $\mathbb{Z}$ recognizing all languages recognized by the syntactic morphism of M .

¹⁴We note that there is also a notion of division in Krebs (2008), but here we link our relational definition directly to language recognition.

Proof. Let $\phi: C \triangleleft M$ be an algebraic division and let $\theta: M \triangleleft S$ be a typed division. Define their composite relation $\psi: C \triangleleft S$ by

$$\psi(\alpha) = \bigcup_{m \in \phi(\alpha)} \theta(m)$$

for each arrow α of C . Because algebraic relational morphisms and divisions compose (Tilson (1987)), ψ is again an algebraic relational morphism and a division from C to the underlying monoid of S .

It remains to check that ψ is also a typed relational morphism. Because M is a one-object category, the hypothesis that θ is typed yields a single ambient set $A \subseteq S$ such that for every $m \in M$ there is a type $\mathfrak{m} \in \mathfrak{T}_S$ with

$$\theta(m) = A \cap \mathfrak{m}.$$

Fix a homset H of C , and put $A_H^\psi := \bigcup_{\beta \in H} \psi(\beta)$. For $\alpha \in H$,

$$\psi(\alpha) = \bigcup_{m \in \phi(\alpha)} (A_H^\psi \cap \mathfrak{m}) = A_H^\psi \cap \left(\bigcup_{m \in \phi(\alpha)} \mathfrak{m} \right).$$

Because M is finite and $\mathfrak{T}_S$ is a Boolean algebra, the finite union

$$\mathfrak{a} := \bigcup_{m \in \phi(\alpha)} \mathfrak{m}$$

is again a type of S . Thus

$$\psi(\alpha) = A_H^\psi \cap \mathfrak{a},$$

so ψ is a typed relational morphism. □

I Algebraic Characterization (Necessary but not Sufficient Criterion)

Theorem 13. $M \in \mathbf{R}^\omega$ iff M is aperiodic and every $\mathcal{R}$ -class of M contains at most one idempotent.

Proof. We note that $\mathbf{R} \circ \mathbf{G} \cap \mathbf{A}$ is exactly the aperiodic monoids with at most one idempotent in each $\mathcal{R}$ -class. First, we show $\mathbf{R}^\omega \subseteq \mathbf{R} \circ \mathbf{G} \cap \mathbf{A}$. Suppose $M \in \mathbf{R}^\omega$. By substituting 1 for y in $\mathbf{R}^\omega$ we obtain that M satisfies

$$x^\omega x = x^\omega$$

which implies $M \in \mathbf{A}$ (Pin, 2009). Next, assume for sake of contradiction that $M \notin \mathbf{R} \circ \mathbf{G}$. Then there is a $\mathcal{R}$ -class of M which contains at least 2 idempotents (Stiffler Jr, 1973, Theorem 3.18). Let $e_1 \neq e_2$ be $\mathcal{R}$ -equivalent idempotents, where $e_1 = e_2 m_2$ and $e_2 = e_1 m_1$. Then

$$(e_1 e_2)^\omega = (e_1 e_1 m_1)^\omega = (e_1 m_1)^\omega = e_2^\omega = e_2$$

while

$$(e_1 e_2)^\omega e_1 = e_2 e_1 = e_2 e_2 m_2 = e_2 m_2 = e_1.$$

This implies $(e_1 e_2)^\omega \neq (e_1 e_2)^\omega e_1$, and thus $M \notin \mathbf{R}^\omega$, a contradiction. Thus $M \in \mathbf{R} \circ \mathbf{G}$.

Now we show $\mathbf{R} \circ \mathbf{G} \cap \mathbf{A} \subseteq \mathbf{R}^\omega$. Let $m_1, m_2 \in M$. First, we note that $(m_1 m_2^\omega)^\omega = (m_2^\omega m_1)^\omega$. This is because

$$\begin{aligned} (m_1 m_2^\omega)^\omega &= (m_1 m_2^\omega)^{\omega+1} && \text{by aperiodicity} \\ &= m_1 (m_2^\omega m_1)^\omega m_2^\omega \\ &\leq_{\mathcal{R}} m_1 (m_2^\omega m_1)^\omega \\ &\leq_{\mathcal{R}} (m_1 m_2^\omega m_1)^\omega m_1 \\ &\leq_{\mathcal{R}} (m_2^\omega m_1)^\omega \end{aligned}$$

and similarly we have $(m_2^\omega m_1)^\omega \leq_{\mathcal{R}} (m_1 m_2^\omega)^\omega$. Then, because $\mathcal{R}$ -equivalent idempotents are equal in $M \in \mathbf{R} \circ \mathbf{G} \cap \mathbf{A}$. Now consider the elements $(m_1 m_2^\omega)^\omega$ and $(m_1 m_2^\omega)^\omega (m_2^\omega m_1)^\omega$, which are both idempotent. We show these are $\mathcal{R}$ -equivalent. First $(m_1 m_2^\omega)^\omega \geq_{\mathcal{R}} (m_1 m_2^\omega)^\omega m_1 \geq_{\mathcal{R}} (m_1 m_2^\omega)^{\omega+1}$, so by aperiodicity $(m_1 m_2^\omega)^\omega \equiv_{\mathcal{R}} (m_1 m_2^\omega)^\omega m_1$. Then

$$\begin{aligned} (m_1 m_2^\omega)^\omega (m_2^\omega m_1)^\omega &= (m_1 m_2^\omega)^\omega m_2^\omega (m_1 m_2^\omega)^\omega m_1 \\ &= (m_1 m_2^\omega)^\omega m_1 m_2^\omega m_2^\omega (m_1 m_2^\omega)^\omega m_1 \\ &= (m_1 m_2^\omega)^\omega m_1 \end{aligned}$$

Since these are in fact both $\mathcal{R}$ -equivalent idempotents, they are equal (Stiffler Jr, 1973). Thus using the above equalities we can conclude

$$(m_1 m_2^\omega)^\omega = (m_1 m_2^\omega)^\omega (m_2^\omega m_1)^\omega = (m_1 m_2^\omega)^\omega m_1.$$

□

Theorem 14. $\mathbf{C-RASP} \cap \mathbf{REG} = \mathbf{wpc}(\mathbf{Dy})$. Hence $\mathbf{R} \subsetneq \mathbf{C-RASP} \cap \mathbf{REG} \subsetneq \mathbf{R}^\omega \subsetneq \mathbf{A} \subsetneq \mathbf{REG}$

Proof. The first claim is shown using the decision procedure (corollary 72): every finite monoid in $\mathbf{C-RASP}$ can be decomposed into wreath products of U_1 and $\mathcal{D}_k$ – in this case $U_1 \preceq \mathcal{D}_1$. In the other direction, $\mathcal{D}_k \in \mathbf{C-RASP}$, and thus $\mathbf{wpc}(\mathbf{Dy})$ contains only monoids in $\mathbf{C-RASP}$, using section E.2. We use this to show the other strict inclusions.

- First, $\mathbf{R} = \mathbf{wpc}(U_1)$. $U_1 \in \mathbf{Dy}$. Then we have that $\mathbf{C-RASP} \cap \mathbf{REG} = \mathbf{wpc}(\mathbf{Dy})$, so $\mathbf{R} \subseteq \mathbf{C-RASP} \cap \mathbf{REG}$. For a strict separation, $\mathcal{D}_1 \in \mathbf{C-RASP} \cap \mathbf{REG}$ but $\mathcal{D}_1 \notin \mathbf{R}$ (Brzozowski & Fich, 1980).
- We have that $\mathbf{C-RASP} \cap \mathbf{REG} = \mathbf{wpc}(\mathbf{Dy})$. Then by proposition 23 we know $\mathcal{D}_k$ divides into a wreath product of U_1 and cyclic groups. We can move all U_1 factors to the left, obtaining a monoid in $\mathbf{R} \circ \mathbf{G}$ (Stiffler Jr, 1973). Furthermore, since all monoids $\mathbf{C-RASP}$ are aperiodic, this results in a monoid in $\mathbf{R}^\omega$, by theorem 13. For a strict separation, the monoid $M((ab + bba)^*) \in \mathbf{R}^\omega \setminus \mathbf{C-RASP}$ (this can be computed by the decision procedure)
- First, $\mathbf{R}^\omega \subseteq \mathbf{A} \subsetneq \mathbf{REG}$ by theorem 13. The strict separation can be witnessed by $M((ab + aabb)^*)$, which is aperiodic but has two idempotents in a single $\mathcal{R}$ -class
- $\mathbf{A} \subsetneq \mathbf{REG}$ is a standard fact. For instance, $M((b^*ab^*ab^*)^*)$ (i.e. $\mathbb{Z}/2\mathbb{Z}$) witnesses the strict separation.

□

J Automata proof of $\mathbf{C-RASP} \cap \mathbf{REG}$

We attempt to track the states of a DFA by a $\mathbf{C-RASP}$ program by iterating over the reachability order of the DFA. Whenever we encounter a nontrivial strongly connected component, we need to check if the program can be extended to cover it. If we succeed for all strongly connected components, the language of the DFA is recognized by a $\mathbf{C-RASP}$ program. Otherwise, we show that failure at any component entails non-membership in $\mathbf{C-RASP}$. First, we will define a property of DFAs which determines their definability in $\mathbf{C-RASP}$. Then, we will argue this property is decidable in polynomial time.

We make the following assumptions, which simplify the presentation of the proof but do not affect the expressivity of $\mathbf{C-RASP}$:

1. All atoms occur bound within $\#$ (unbound atoms have a constant truth value since we would evaluate them on an $\langle \text{EOS} \rangle$ token).
2. No atom appears negatively (since atoms are disjoint, we didn't need negation on them anyways).

Throughout, write $\mathcal{A} = (Q, \delta, \Sigma)$ for a semiautomaton where δ is extended to Σ^* as usual. We write $t^{q,w,i}$ for the value of a C-RASP term t at position i of w when the run starts in state q , and $q, w, i \models \phi$ for the truth of a formula at that position. We write $t^{q,w}$ as shorthand for $t^{q,w,|w|}$. We write $L(\mathcal{A}, q_0, F)$ to denote the language recognized by $\mathcal{A}$ with start state $q_0 \in Q$ and accepting states $F \subseteq Q$.

Definition 78 (SCC). $W \subseteq Q$ is a strongly connected component of $\mathcal{A}$ if it is maximal such that for all $q_1, q_2 \in W$ there is $w \in \Sigma^*$ with $\delta(q_1, w) = q_2$. W is trivial if $W = \{q\}$ and $\delta(q, \sigma) \neq q$ for every σ .

We will be reasoning about individual SCCs of an automaton, so we formalize what it means to take an SCC out of an automaton.

Definition 79 (Extraction). Let W be an SCC of $\mathcal{A}$. The extraction of W is the semiautomaton $\mathcal{A} \upharpoonright W = (W \sqcup S_W, \delta_W, \Sigma)$ where

$$S_W = \{\delta(p, \sigma) \mid p \in W, \sigma \in \Sigma\} \setminus W$$

$$\delta_W(q, \sigma) = \begin{cases} \delta(q, \sigma) & q \in W \\ q & q \in S_W. \end{cases}$$

In other words, to extract an SCC you isolate the states of W and all other states reachable in one transition. These external states become sink states in the extracted semiautomaton. If a semiautomaton $\mathcal{A}$ is the extraction of a SCC from itself, then we say $\mathcal{A}$ is strongly connected. In the sequel, we will denote by S_W the external sink states of an extracted SCC. We define the following helper function, which acts differently on internal states and external sink states of an SCC.

Definition 80. Let $\mathcal{A}$ be a semiautomaton, W a SCC, q_0 a state of $\mathcal{A}$, and t a term of C-RASP. We define the following function:

$$V_{q_0, q}(t) = \begin{cases} \{t^{q_0, w, i} \mid \delta_W(q_0, w_{\leq i}) = q\} & q \in W \\ \{t^{q_0, w, i} \mid \delta_W(q_0, w_{\leq i-1}) \in W, \delta_W(q_0, w_{\leq i}) = q\} & \text{otherwise} \end{cases}$$

In other words, for states in W , the function $V_{q_0, q}$ take the set of all realizable counts on a path from $q_0 \rightarrow q$, and for states immediately outside of W we take the counts realizable by a path $q_0 \rightarrow q$ which takes a single step from W to q . We will distinguish states in an automaton using terms which realize distinct sets of counts when in each state. However, only certain sets of counts can be distinguished by C-RASP formulas (which can be formalized by the “types” as in section E.2).

Definition 81 (Separated). Let W be an SCC. Distinct $q_1, q_2 \in \mathcal{A} \upharpoonright W$ are separated by a term t if there is some $q_0 \in W$ such that $V_{q_0, q_1}(t) \cap V_{q_0, q_2}(t) = \emptyset$. We say W is separable if every pair of distinct states in $\mathcal{A} \upharpoonright W$ is separated by some t . We say W is separable if every pair of distinct states in $\mathcal{A} \upharpoonright W$ is separated by some t .

Definition 82 (Balanced). The subterms of a term t are t itself alongside every term t' occurring inside a subformula $[t' \geq C]$. A term t is balanced on W if there is an interval $[\alpha, \beta]$ such that $V_{q_0, q}(t) \subseteq [\alpha, \beta]$ for all $q_0, q \in W$. A term t is totally balanced on W if all subterms $t' \in t$ are balanced on W . A formula ϕ is balanced on W if every term occurring in ϕ is balanced on W .

Intuitively, if ϕ is not balanced on W then we would see arbitrarily large or small values when checking the counters when the DFA is in states in W . An important observation is that for balanced terms, $|V(t)| = 1$.

Lemma 83. If t is balanced on W , then $|V_{q_0, q}(t)| = 1$ for all $q_0, q \in Q$.

Proof. Suppose otherwise that some $V_{q_0, q}(t)$ holds two values $x_1 \neq x_2$. Since cycles in W must sum to 0 (otherwise t would be unbalanced), $V_{q, q_0}(t)$ must contain $-x_1$ and $-x_2$. However, this implies the existence of some cycle $q_0 \rightarrow q_0$ with weight $x_1 - x_2 \neq 0$. $\square$

With these notions at hand, we are ready to state the main claim.

Lemma 84 (Definability Criterion). *Let $\mathcal{A} = (Q, \delta, \Sigma)$ be a semiautomaton. The following are equivalent*

1. $L(\mathcal{A}, q_0, F)$ is recognizable in C-RASP for any $q_0 \in Q$ and $F \subseteq Q$.
2. $\mathcal{A} \upharpoonright W$ is separable by totally balanced terms for every SCC W in $\mathcal{A}$.

Proof. First, (2) $\Rightarrow$ (1) is shown in lemma 84. Then, $\neg(2) \Rightarrow \neg(1)$ is shown in lemma 84. $\square$

Lemma 85. [Construction] *Suppose every SCC in $\mathcal{A}$ is separable by totally balanced terms. Fix $q_0 \in Q$. Then for each $q \in Q$ there exists a C-RASP formula ϕ_q such that $w \models \phi_q \iff \delta(q_0, w) = q$.*

Proof. First, for any $q \in Q$ not reachable from q_0 we can set $\phi_q = \perp$. Otherwise, we induct on the reachability order of SCC's in $\mathcal{A}$. Consider the first SCC W containing q_0 . Because W is separable by a totally balanced terms, for each $q_1, q_2 \in \mathcal{A} \upharpoonright W$ there exists a totally balanced term t that separates them. We will define a formula $\phi_{q_1, \neg q_2}$ such that strings which land on q_1 always satisfy $\phi_{q_1, \neg q_2}$ while strings that land in q_2 do not. Since the sets of counts upon landing in each state form two disjoint finite sets, we can check these counts using a C-RASP formula:

$$\phi_{q_1, \neg q_2} := \left(\bigvee_{c \in V_{q_0, q_1}(t)} t = c \right) \wedge \neg \left(\bigvee_{c \in V_{q_0, q_2}(t)} t = c \right)$$

Then, we can let $\phi_q := \bigwedge_{q' \neq q \in \mathcal{A} \upharpoonright W} \phi_{q, \neg q'}$. For the inductive step on an SCC W' , we have by assumption formulas $\phi_{q'_0}$ which detect entry into W' (as these are states in S_W from previous SCC's). By assumption, there exists $\phi_{q'_0, q}$ which detects if $\mathcal{A}$ is in q if we started in state $\phi_{q'_0}$. Since, by assumption, no atom appears negatively, we can take ϕ_q and apply the mapping $\sigma \mapsto (\sigma \wedge \# \phi_{q'_0} \geq 1)$ to transform $\phi_{q'_0, q} \mapsto \hat{\phi}_{q'_0, q}$. Intuitively, this transformation tells $\phi_{q'_0, q}$ to ignore all symbols which occurred before the first q'_0 (i.e. before we entered W' via q'_0). Now, define the set of entry points $E_{W'} = \{q \mid \delta(q', \sigma) = q, q' \notin W'\}$, and then we can let $\phi_q = \bigvee_{q'_0 \in E_{W'}} \hat{\phi}_{q'_0, q}$. $\square$

Lemma 86 (Contradiction). *Let $\mathcal{A}, q_0, F$ define the minimal automaton of a language. Suppose there exists an SCC W in $\mathcal{A}$ which is not separable by any totally balanced terms. Then $L(\mathcal{A}, q_0, F)$ is not definable in C-RASP.*

Proof. Let ϕ be any C-RASP formula and C_0 be larger than any threshold or coefficient in any subformula of the form $\sum \chi \cdot \#[\psi] \geq C$. Since W is not separable by totally balanced terms, there are $q_1, q_2 \in W$ such that for all totally balanced terms t and entry points $\delta(q_0, x) = q_i$, we have that $V_{q_i, q_1}(t) = V_{q_i, q_2}(t)$. Let w_1 and w_2 be the minimal strings such that $\delta(q_i, w_1) = q_1$ and $\delta(q_i, w_2) = q_2$. Let $v \in \Sigma^*$ be any distinguishing suffix of q_1 and q_2 – such that $\delta(q_1, v) \in F$ while $\delta(q_2, v) \notin F$ – which always exists as q_1, q_2 are distinct states in the minimal automaton. We will prepend very large loops around q_i , carefully chosen so that all unbalanced terms $\gg C_0$ or $\ll C_0$.

Order all unbalanced terms $t_1, t_2, \dots, t_k$. First, we note that for all unbalanced t_i there exists some loop $u_i \in \Sigma^*$ such that $\delta(q_i, u_i) = q_i$ and the loop sum $t_i^{q_i, u_i u_i} - t_i^{q_i, u_i} \neq 0$ – otherwise, $V_{q_i, q_i}(t_i)$ would be bounded and t_i would not be unbalanced on W . Then, there exists some constant c such that the loop $u := u_1^c u_2^{c-1} \dots u_k^c$ has a nonzero sum for all terms (The constant is chosen such that no subsequent loop can “undo” the counts accumulated in previous loops, thus guaranteeing that all terms will have a nonzero sum).

There exists a sufficiently large exponent N such that over a long prefix xu^N , replacing all unbalanced terms with ∞ (or $-\infty$, depending on the sum over u) results in a formula ψ whose truth value matches ϕ on all positions after the prefix in $xu^N w_1$ and $xu^N w_2$. Intuitively, any formula $[t \geq C]$ converges to $\top$ or $\perp$ after enough iterations of u , and the

remaining iterations are used to drown out the finite prefix before convergence. Now, ψ had all unbalanced subformulas replaced with constants, so it is totally balanced. Thus, $tq_0, xu^N w_1 = tq_0, xu^N w_2$, and appending the suffix v does not change this, so ϕ will have the same truth value on $xu^N w_1 v$ as $xu^N w_2 v$ – therefore ϕ cannot define $L(\mathcal{A}, q_0, F)$. $\square$

Now we will show that these conditions are decidable in polynomial time. We say that two terms are equivalent up to separation of states in the following sense: Each term defines an equivalence relation $\equiv_{t, q_0}$ over states, where $q_1 \approx_{t, q_0} q_2$ whenever $V_{q_0, q_1}(t) = V_{q_0, q_2}(t)$.

Lemma 87. [Depth-1 Balanced] Let $\mathcal{A} = (Q, \delta, \Sigma)$ be an SCC. The set of all depth-1 C-RASP terms up to equivalent separability of states can be computed in $O(\text{poly}(|Q|, |\Sigma|))$ time.

Proof. Let $q_0 \in Q$. For every term $\sum \chi_p \# [\phi_p]$ occurring in a balanced depth-1 we must have that $(\sum \chi_\sigma \# [\sigma])^{w, |w|} = 0$ over any loop w , i.e. when $\delta(q_0, w) = q_0$. Here, we use the fact that balanced terms are single-valued on each state of an SCC (lemma 83). First, we assert that such a balanced term exists iff there exists a function $E: \Sigma \rightarrow \mathbb{Z}$ and $V: Q \rightarrow \mathbb{Z}$ such that the following condition is satisfied:

$$\text{If } \delta(q_1, \sigma) = q_2 \text{ then } V(q_2) = V(q_1) + E(\sigma) \quad (23)$$

If such a function exists, we can obtain a term that sums to 0 on loops via $\sum E(\sigma) \# [\sigma]$. If such a term $\sum \chi_\sigma \# [\sigma]$ exists, we can obtain the function by setting $E(\sigma) = \chi_\sigma$ and $V(q) = \sum_{1 \leq i \leq |w|} E(w_i)$ for any w such that $\delta(q_0, w) = q$. The only other balanced terms are linear combinations of these terms, but these would all be equivalent with respect to separability of states. Deciding the existence of such a function can be done in $O(\text{poly}(|Q|, |\Sigma|))$ time. There are $O(|Q|^2 |\Sigma|)$ many transitions $\delta(q_1, \sigma) = q_2$ so we obtain $O(|Q|^2 \cdot |\Sigma|)$ many constraints of the form $V(q_2) = V(q_1) + E(\sigma)$. These give $O(|Q|^2 |\Sigma|)$ -many linear constraints over $O(|Q|^2 \cdot |\Sigma| + |Q|)$ -many variables. This can be solved in $O(\text{poly}(|Q|, |\Sigma|))$ time (Kannan & Bachem, 1979). $\square$

Lemma 88. Whether an SCC W is separable by totally balanced terms is decidable in $O(\text{poly}(|Q|, |\Sigma|))$ time.

Proof. We will iteratively refine an equivalence relation $\equiv$ over states, based upon the set of $V_{q_0, q}(t)$ over all terms. At the end, if all states are in their own equivalence class, the SCC is separable by a balanced formula.

1. Initialize $\equiv_0 := W \times W$ by setting all elements to the same class. Initialize a set of C-RASP formulas $\Phi_0 := \emptyset$. Fix a start state q_0 (since a balanced term is balanced given any start state, the exact choice is immaterial). Then perform the loop (2) – (4):
2. Generate the set B_k of all terms that count over formulas in Φ_{k-1} , up to equivalent separability of states using lemma 87.
3. Denote the indicator for $K \subseteq W$ induced by t as the formula

$$\psi_K := \bigvee_{q \in K} t = V_{q_0, q}.$$

We generate the collection of indicators of separable state sets by all $t \in B_k$ as the collection $\Psi_k := \{\psi_K \mid V_{q_0, q_1}(t) = V_{q_0, q_2}(t) \text{ for all } q_1, q_2 \in K\}$.

4. Now there are two cases

- If $\Psi_k \not\subseteq \Phi_{k-1}$, then define Φ_k to refine Φ_{k-1} via intersection with Ψ_k .

$$\Phi_k := \{\psi_{K_1} \cap \psi_{K_2} \mid \psi_{K_1} \in \Phi_{k-1}, \psi_{K_2} \in \Psi_k\}.$$

Next, set $\equiv_k := \{(q_1, q_2) \mid q_1, q_2 \in K \text{ for some } \phi_K \in \Phi_k\}$. Then, return to step (2) for iteration $k + 1$.

- If $\Psi_k \subseteq \Phi_{k-1}$, then we have reached a fixed point. Set $\Phi := \Phi_k$ and $\equiv := \equiv_k$ and move to step (5).

5. If $[q]_{\equiv}$ is a singleton for all q , output TRUE and return Φ . Otherwise, output FALSE.

Correctness: Since each $\phi_K \in \Phi$ is balanced, all its subterms are balanced and thus if $[q]_{\equiv}$ are singletons we can obtain totally balanced terms which separate each pair of states. If some $[q]_{\equiv}$ is not a singleton, we know there are two states that no totally balanced term can separate. Thus, the algorithm is correct.

Polynomial running time: We crucially maintain that $\equiv_k$ is monotonically refined at each step, and Φ keeps only the formulas that define equivalence classes via $\equiv_k$, and thus there are only $O(|Q|^2)$ many of them at each iteration. The first step (1) runs in $O(|Q|^2)$ time. Step (2) takes $O(\text{poly}(|Q|, |\Sigma|))$ time by lemma 87. Step (3) runs in $O(\text{poly}(|Q|, |\Sigma|))$ time by generating in $O(|Q|^2)$ time the equivalence class where $V_{q_0, q_1}(t) = V_{q_0, q_2}(t)$ for each of the $O(\text{poly}(|Q|, |\Sigma|))$ terms $t \in B$. Step (4) runs in $O(\text{poly}(|Q|, |\Sigma|))$ time, since the inclusion check is over polynomially sized sets and the intersection only needs to be done via pairs in $\Phi_{k-1} \times \Psi_k$, since each formula identifies a disjoint set of states K . Step (5) also just needs to check if the equivalence relation has size $|Q|$, achievable in $O(|Q|)$ time. Finally, we only need to iterate the (2 – 4) loop $O(|Q|^2)$ many times, as the equivalence relation $\equiv_k$ is strictly refined at each step until a fixed-point is reached. Thus, this algorithm runs in polynomial time. $\square$

Theorem 89. *Checking whether or not every SCC in $\mathcal{A}$ is separable by a totally balanced terms is decidable in time polynomial in the size of $\mathcal{A}$.*

Proof. We iterate over $O(\text{poly}(|Q|, |\Sigma|))$ many SCCs (e.g. using Tarjan’s algorithm (Tarjan, 1972)), and only need to perform a $O(\text{poly}(|Q|, |\Sigma|))$ time query on each one, as in lemma 88. $\square$

K Implementation

A sketch of the algorithm used in the python implementation we have provided. For simplicity the implementation uses a looser version the algorithm which enumerates all loops in a strongly conected component (of which there may be exponentially many), though one could modify it to strictly be a polynomial time algorithm, as proven above. For each simple loop in a strongly connected component, the helper function BALANCEDLABELS-BASIS computes the basis of all morphisms into $\mathbb{Z}$ given the constraint that all loops must be “balanced” (i.e. have the images of the symbols sum to 0). By computing the nullspace of this basis, we find all possible morphisms that sum to 0 on loops, and iteratively relable the symbols according to these morphisms. At the end, the helper function SEPARATED checks if all states have differing sets of outgoing transition labels (hence the states can be distinguished only using iterated counting – and thus being expressible in C-RASP).

Algorithm 1 C-RASP Membership

```

1: procedure DECIDEMEMBERSHIP( $\mathcal{A}$ )
2:   SCCs  $\leftarrow$  STRONGLYCONNECTEDCOMPONENTS( $\mathcal{A}$ )
3:   for all SCC  $\in$  SCCs do
4:      $B_{\text{prev}} \leftarrow \emptyset$ 
5:      $B_{\text{curr}} \leftarrow$  BALANCEDLABELSBASIS(SCC)
6:     if NULLSPACE( $B_{\text{curr}}$ )  $\neq \emptyset$  then
7:       while  $B_{\text{curr}} \notin \text{SPAN}(B_{\text{prev}})$  do
8:          $\text{labels}[\sigma_i] \leftarrow [v_i]_{v \in B_{\text{curr}}}$ 
9:         for all  $\text{loop} \in \text{LOOPSAROUND}(q_0, \text{SCC})$  do
10:          for all  $0 \leq i \leq \text{LEN}(\text{loop})$  do
11:             $(q_i, \sigma_i, q_{i+1}) \leftarrow \text{loop}[i]$ 
12:             $\text{SCC}.\delta.\text{REPLACE}\left((q_i, \sigma, q_{i+1}), \left(q_i, \left[\sigma_i, \sum_{1 \leq j \leq i} \text{labels}[\sigma_j]\right], q_{i+1}\right)\right)$ 
13:          end for
14:           $B_{\text{prev}} \leftarrow B_{\text{curr}}$ 
15:           $B_{\text{curr}} \leftarrow$  BALANCEDLABELSBASIS(SCC)
16:        end for
17:      end while
18:    end if
19:    if  $\neg \text{SEPARATED}(\text{SCC})$  then
20:      return  $\perp$ 
21:    end if
22:  end for
23:  return  $\top$ 
24: end procedure

```
