Characterizing NC^1 with Typed Monoids

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, UK

Aidan T. Evans[1]
Department of Computer Science and Technology, University of Cambridge, UK

Abstract

Krebs et al. (2007) gave a characterization of the complexity class TC^0 as the class of languages recognized by a certain class of typed monoids. The notion of typed monoid was introduced to extend methods of algebraic automata theory to infinite monoids and hence characterize classes beyond the regular languages. We advance this line of work beyond TC^0 by giving a characterization of NC^1. This is obtained by first showing that NC^1 can be defined as the languages expressible in an extension of first-order logic using only unary quantifiers over regular languages. The expressibility result is a consequence of a general result showing that finite monoid multiplication quantifiers of higher dimension can be replaced with unary quantifiers in the context of interpretations over strings, which also answers a question of Lautemann et al. (2001). We estblish this collapse result for a much more general class of interpretations using results on interpretations due to Bojańczyk et al. (2019), which may be of independent interest.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Complexity theory and logic; Theory of computation → Circuit complexity; Theory of computation → Finite Model Theory

Keywords and phrases algebraic automata theory, circuit complexity, descriptive complexity, typed monoids, semigroups, generalized quantifiers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.26

Related Version *Full Version*: `https://arxiv.org/abs/2508.11019` [8]

Funding *Anuj Dawar*: Research funded in part by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee: grant number EP/X028259/1.
Aidan T. Evans: Researched supported by Huawei HiSilicon Studentship.

1 Introduction

Much work in theoretical computer science is concerned with studying classes of formal languages, whether these are classes defined in terms of grammars and expressions, such as the class of regular or context-free languages, or whether they are *complexity classes* such as P and NP, defined by resource bounds on machine models. Indeed, the distinction between these is largely historical as most classes of interest admit different characterizations based on machine models, grammars, logical definability, or algebraic expressions. The class of regular languages can be characterized as the languages accepted by linear-time-bounded single-tape Turing machines [12] while P can be characterized without reference to resources as the languages recognized by multi-head two-way pushdown automata [7]. The advantage of the variety of characterizations is, of course, the fact that these bring with them different mathematical toolkits that can be brought to the study of the classes.

The class of regular languages has arguably the richest theory in this sense of diversity of characterizations. Most students of computer science learn of the equivalence of deterministic and nondeterministic finite automata, regular languages and linear grammars and many

[1] Corresponding Author

also know that the regular languages are exactly those definable in monadic second-order logic with an order predicate. Perhaps the most productive approach to the study of regular languages is via their connection to finite monoids. Every language L has a syntactic monoid, which is finite if, and only if, L is regular. Moreover, closure properties of classes of regular languages relate to natural closure properties of classes of monoids, via Eilenberg's Correspondence Theorem [11]. Finally, we have *Krohn-Rhodes theory* which decomposes regular languages into elementary components; such a decomposition has found applications not only in semigroup theory but also, for example, in physics [17, Sections 4.8a, 4.10] and the modeling of biochemical reactions [17, Chapter 6, Part I]. All together, these tools give rise to *algebraic automata theory* – which leads to the definition of natural subclasses of the class of regular languages, to effective decision procedures for automata recognizing such classes, and to separation results.

When it comes to studying computational complexity, we are mainly interested in classes of languages richer than just the regular languages. Thus, the syntactic monoids of the languages are not necessarily finite and the extensive tools of Krohn-Rhodes theory are not available to study them. Nonetheless, some attempts have been made to extend the methods of algebraic automata theory to classes beyond the regular languages. Most significant is the work of Krebs and collaborators [2, 3, 14, 13, 6], which introduces the notion of *typed monoids*. The idea is to allow for languages with infinite syntactic monoids, but limit the languages they recognize by associating with the monoids a finite collection of types. This allows for the formulation of a version of Eilenberg's Correspondence theorem associating closure properties on classes of typed monoids with corresponding closure properties of classes of languages. In particular, this implies that most complexity classes of interest can be uniquely characterized in terms of an associated class of typed monoids [3]. An explicit description of the class characterizing uniform TC^0 is given in Krebs et al. [14, 13] and, furthermore, provides a decomposition of TC^0 into elementary components analogous to the Krohn-Rhodes theory for regular languages. This is obtained through a general method which allows us to construct typed monoids corresponding to *unary quantifiers* defined from specific languages [13] (see also Theorem 11 below).

In this paper, we extend this work to obtain a characterization and decomposition of (uniform) NC^1 as the class of languages recognized by the collection of typed monoids obtained as the closure under *ordered strong block products* of three typed monoids: the group of integers with types for positive and non-positive integers; the monoid of the natural numbers with types for the square numbers and non-square numbers; and a finite non-solvable group such as S_5 with a type for each subset of the group. Full definitions of these terms follow below. Our result is obtained by first characterizing NC^1 in terms of logical definability in an extension of first-order logic with only unary quantifiers. It is known that any regular language whose syntactic monoid is a non-solvable group is complete for NC^1 under reductions definable in first-order logic with arithmetic predicates ($(\mathrm{FO})[+,\times]$) [1]. From this, we know we can describe NC^1 as the class of languages definable in an extension of $(\mathrm{FO})[+,\times]$ with quantifiers (of arbitrary arity) associated with the regular languages recognized by the monoid S_5. We show that the family of such quantifiers associated with any finite monoid can be replaced with just the unary quantifiers. This answers a question left open in Lautemann et al. [16] and allows us to obtain the sought after algebraic characterization.

For the purpose of establishing the characterization of NC^1, it suffices to consider quantifiers applied to interpretations in which tuples of elements are ordered lexicographically. However, we show that the arity collapse of quantifiers associated with finite monoids holds more generally, for any first-order definable linear orders on tuples. To show this, we leverage a characterization of first-order interpretations due to Bojańczyk et al. [5, 4].

We begin in Section 2, by covering relevant background on semigroup theory, typed monoids, and multiplication quantifiers (some of which is relegated to the appendix in the interests of space). In Section 3, we establish the technical result showing that quantifiers of higher arity over a regular language L can be defined using just unary quantifiers over the syntactic monoid of L, when the quantifiers are applied to interpretations with a lexicographic order on tuples. In Section 4, we apply this to obtain the algebraic characterization of NC^1. finally, in Section 5, we establish the arity collapse for quantifiers in the context of more general interpretations. A full version of this paper with a detailed background is available on arXiv [8].

2 Preliminaries

We assume the reader is familiar with basic concepts of formal language theory, automata theory, complexity theory, and logic. We quickly review definitions we need in order to fix notation and establish conventions.

We write $\mathbb{Z}$ for the set of integers, $\mathbb{N}$ for the set of natural numbers (including 0), and $\mathbb{Z}^+$ for the set of positive integers. We write $[n]$ for the set of integers $\{1, \ldots, n\}$ and $\mathbb{S}$ for the set of *square* integers. That is, $\mathbb{S} = \{x \in \mathbb{Z}^+ \mid x = y^2 \text{ for some } y \in \mathbb{Z}\}$.

For a fixed $n \in \mathbb{Z}^+$ and an integer $i \in [n]$, we define the *n-bit one-hot encoding* of i to be the binary string $b \in \{0,1\}^n$ such that $b_j = 1$ if, and only if, $j = i$. For any set S, we write $\wp(S)$ for the powerset of S.

The complexity classes TC^0 and NC^1 are classes of languages defined in terms of circuits. We are only interested in the *uniform* versions of these classes. Specifically, TC^0 is the class of languages recognized by a uniform family of circuits of polynomial size and constant depth using And, Or, Not and Majority gates of arbitrary fan-in; and NC^1 is the class of languages recognized by a uniform family of circuits of polynomial size and logarithmic depth using And, Or and Not gates of fan-in at most 2. Since gates (including majority gates) of unbounded fan-in can be simulated by circuits of logarithmic depth of fan-in 2 using just the standard Boolean basis, we have $\mathrm{TC}^0 \subseteq \mathrm{NC}^1$ and the separation of the two classes is open. The requirement of *uniformity* here means that the circuits of input size n are easily computed from n. It is standard to take "easily computed" to mean DLogTime-uniform, though the classes are robust under varying the definition (see [20]).

2.1 Semigroups, Monoids, and Groups

A *semigroup* $(S, \cdot)$ is a set S equipped with an *associative* binary operation. We call a semigroup *finite* if S is finite. Context permitting, we may refer to a semigroup $(S, \cdot)$ simply by its underlying set S.

A *monoid* $(M, \cdot)$ is a semigroup with a distinguished element $1_M \in M$ such that for all $m \in M$, $1_M \cdot m = m \cdot 1_M = m$. We call 1_M the *identity* or *neutral* element of M. A *group* $(G, \cdot)$ is a monoid such that for every $g \in G$, there exists an element $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = 1$. We call g^{-1} the *inverse* of g.

Note that $(\mathbb{Z}, +)$ is a group, $(\mathbb{N}, +)$ is a monoid but not a group and $(\mathbb{Z}^+, +)$ is a semigroup but not a monoid. In the first two cases, the identity element is 0. When we refer to the monoids $\mathbb{Z}$ or $\mathbb{N}$ we assume that the operation referred to is standard addition.

For a semigroup $(S, \cdot)$, we say that a set $G \subseteq S$ *generates* S if S is equal to the closure of G under $\cdot$; we denote this by $S = \langle G \rangle$., or, simply, $\langle G \rangle$ if the operation is clear from context, and call G a *generating set of* S. We say that S is *finitely generated* if there exists a finite generating set of S. All semigroups we consider are finitely generated. Note that $\mathbb{Z}^+$ is generated by $\{1\}$, $\mathbb{N}$ by $\{0, 1\}$ and $\mathbb{Z}$ by $\{1, -1\}$.

We write U_1 for the monoid $(\{0,1\},\cdot)$ where the binary operation is the standard multiplication. Note that 1 is the identity element here. For any set S, we denote by S^+ the set of non-empty finite strings over S and by S^* the set of all finite strings over S. Equipped with the concatenation operation on strings, which we denote by either $\circ$ or simply juxtaposition, S^* is a monoid and S^+ is a semigroup but not a monoid. We refer to these as the *free monoid* and *free semigroup* over S, respectively. Note that S is a set of generators for S^+ and $S \cup \{\epsilon\}$ is a set of generators for S^*.

A monoid homomorphism from a monoid $(S,\cdot_S)$ to a monoid $(T,\cdot_T)$ is a function $h : S \to T$ such that for all $s_1, s_2 \in S$, $h(s_1 \cdot_S s_2) = h(s_1) \cdot_T h(s_2)$ and $h(1_S) = 1_T$. A *congruence* on a monoid $(M,\cdot)$ is an equivalence relation $\sim$ on M such that for all $a,b,c,d \in M$, if $a \sim b$ and $c \sim d$, then $a \cdot c \sim b \cdot d$. We denote by $M/\!\sim$ the set of equivalence classes of $\sim$ on M. We denote by $[a]_\sim$, or simply $[a]$, the equivalence class of $a \in M$ under $\sim$. Any congruence $\sim$ gives rise to the *quotient monoid* of M by $\sim$, namely the monoid $(M/\!\sim,\star)$ where for $[a],[b] \in M/\!\sim$, $[a] \star [b] = [a \cdot b]$. The map $\eta : M \to M/\!\sim$ defined by $\eta(a) = [a]$ is then a homomorphism, known as the *canonical homomorphism* of M onto $M/\!\sim$.

2.2 Language Recognition with Monoids

Here we recall the definition of syntactic congruences and syntactic monoids.

For a language $L \subseteq \Sigma^*$ over an alphabet Σ, we define the *syntactic congruence* of L as the equivalence relation $\sim_L$ on Σ^* such that for all $x,y \in \Sigma^*$, $x \sim_L y$ if, and only if, for all $w,v \in \Sigma^*$, $wxv \in L$ if, and only if, $wyv \in L$.

It is easily seen that this relation is a congruence on the free monoid Σ^*. The quotient monoid $\Sigma^*/\!\sim_L$ is known as the *syntactic monoid* of L. More generally, we say that a monoid M *recognizes* a language L if there is a homomorphism $h : \Sigma^* \to M$ and a set $A \subseteq M$ such that $L = h^{-1}(A)$. It is easily seen that the syntactic monoid of L recognizes L. A language is regular if, and only if, its syntactic monoid is finite [19].

2.3 Logics and Quantifiers

We assume familiarity with the basic syntax and semantics of first-order logic. In this paper, the logic is always interpreted in finite relational structures. We generally denote structures by Fraktur letters, $\mathfrak{A}$, $\mathfrak{B}$, etc., and the corresponding universe of the structure is denoted $|\mathfrak{A}|$, $|\mathfrak{B}|$, etc. We are almost exclusively interested in *strings* over a finite alphabet. Thus, fix an alphabet Σ. A Σ-string is then a structure $\mathfrak{A}$ whose universe $|\mathfrak{A}|$ is linearly ordered by a binary relation $<$ and which interprets a set of unary relation symbols $(R_\sigma)_{\sigma\in\Sigma}$ such that for each element $a \in |\mathfrak{A}|$ there is a unique $\sigma \in \Sigma$ such that a is in the interpretation of R_σ.

More generally, let τ be any relational vocabulary consisting of a binary relation symbol $<$ and unary relation symbols $R_1,\ldots,R_k$. We can associate with any τ-structure in which $<$ is a linear order a string over an alphabet of size 2^k as formalized in the following definition.

▶ **Definition 1.** *For τ a relational vocabulary consisting of a binary relation symbol $<$ and unary relation symbols $R_1,\ldots,R_k$, and $\mathfrak{A}$ a τ-structure with n elements that interprets the symbol $<$ as a linear order of its universe, we define the string $w_\mathfrak{A}$* associated with *$\mathfrak{A}$ as the string of length n over the alphabet $\Sigma_k = \{0,1\}^k$ of size 2^k so that if a is the ith element of $w_\mathfrak{A}$, then a is the k-tuple where $a_j = 1$ if, and only if, R_j holds at the ith element of $\mathfrak{A}$.*

For example, say $\tau = \{R_1,R_2,R_3\}$ and $\mathfrak{A} = ([4],<^\mathfrak{A},R_1^\mathfrak{A},R_2^\mathfrak{A},R_3^\mathfrak{A})$ where $1 < 2 < 3 < 4$, $R_1^\mathfrak{A} = \{1,3\}$, $R_2^\mathfrak{A} = \{2,4\}$, and $R_3^\mathfrak{A} = \{1,2,3\}$. Then, the string $w_\mathfrak{A} = (101)(011)(101)(010)$.

Using the above, we can associate a language with any isomorphism-closed class of structures over the vocabulary τ. We formalize this definition for future use.

▶ **Definition 2.** *For τ a relational vocabulary consisting of a binary relation symbol $<$ and unary relation symbols $R_1, \ldots, R_k$, and $\mathcal{A}$ a class of τ-structures where every structure interprets $<$ as a linear order, we define the language $L_{\mathcal{A}}$ over the alphabet $\Sigma_k = \{0,1\}^k$ to be*

$$L_{\mathcal{A}} = \{w_{\mathfrak{A}} \mid \mathfrak{A} \in \mathcal{A}\}.$$

Conversely, for any language L over the alphabet Σ_k, we define the class of τ-structures $\mathcal{S}_L$ to be

$$\mathcal{S}_L = \{\mathfrak{A} \mid w_{\mathfrak{A}} \in L\}.$$

It is worth noting that the above operations are not inverses, in the sense that it is not the case that $S_{L_{\mathcal{A}}} = \mathcal{A}$. This is because while $\mathcal{A}$ is a class of structures using k unary relation symbols, $S_{L_{\mathcal{A}}}$ is in a vocabulary using 2^k unary symbols.

As the elements of a string $\mathfrak{A}$ are linearly ordered, we can identify them with an initial segment $\{1, \ldots, n\}$ of the positive integers. In other words, we treat a string with universe $\{1, \ldots, n\}$ and the standard order on these elements as a canonical representative of its isomorphism class. In addition to the order predicate, we may allow other *numerical predicates* to appear in formulas of our logics. These are predicates whose meaning is completely determined by the size n of the structure and the ordering of its elements. In particular, we have ternary predicates $+$ and $\times$ for the partial addition and multiplication functions.

An insight due to Lindström allows us to define a *quantifier* from any isomorphism-closed class of structures (see [10]). We adopt the terminology of Ebbginhaus and Flum [9, Chapter 12] and provide a brief overview now. Consider a relational vocabulary $\tau = \{R_1, \ldots, R_l\}$, where for each i, R_i is a relation symbol of arity r_i. For any vocabulary σ and positive integer d, an *interpretation* of τ in σ of dimension d is a tuple of formulas $I = (\phi_1(\overline{x}_1), \ldots, \phi_l(\overline{x}_l))$ of vocabulary σ where ϕ_i is associated with a tuple $\overline{x}_i$ of variables of length $|\overline{x}_i| = dr_i$. Suppose we are given a σ-structure $\mathfrak{A}$ and an assignment α that takes variables to elements of $\mathfrak{A}$. Then let $\phi_i^{\mathfrak{A},\alpha}$ denote the relation of arity dr_i consisting of the set of tuples $\{\overline{a} \in |\mathfrak{A}|^{dr_i} \mid \mathfrak{A} \models \phi_i[\alpha[\overline{x}_i/\overline{a}]]\}$. Then, the interpretation I defines a map that takes a σ-structure $\mathfrak{A}$, along with an assignment α to the τ-structure $I(\mathfrak{A}, \alpha)$ with universe $|\mathfrak{A}|^d$ where the interpretation of R_i is the set $\phi_i^{\mathfrak{A},\alpha}$, seen as a relation of arity r_i on $|\mathfrak{A}|^d$.

Given a class of τ-structures Q and any positive integer d, we have a quantifier Q_d. In a logic with Q_d, we can form formulas of the form

$$Q_d \overline{x}_1 \cdots \overline{x}_l(\phi_1, \ldots, \phi_l)$$

whenever $I = (\phi_1(\overline{x}_1), \ldots, \phi_l(\overline{x}_l))$ is an interpretation of dimension d. In this formula, occurrences in the subformula ϕ_i of variables among x_i are bound. The semantics of this quantifier are given by the rule that $Q_d \overline{x}_1 \cdots \overline{x}_l(\phi_1, \ldots, \phi_l)$ is true in a structure $\mathfrak{A}$ under some assignment α of values to the free variables if the τ-structure $I(\mathfrak{A}, \alpha)$ is in Q. We can understand Q_d as the dth vectorization of the quantifier Q (see [9, Def. 12.1.6]). When $d = 1$, we may omit the subscript. A quantifier of the form Q_1 is called *unary*.

The standard first-order quantifiers: $\exists$ and $\forall$ can be seen as special cases of Lindström quantifiers in a vocabulary with one unary relation U. The existential quantifier consists of all structures (A, U) where $U \subseteq A$ is non-empty and the universal quantifier consists of all structures (A, U) where $U = A$.

We are particularly interested in interpretations I where both σ and τ are vocabularies of strings. These are also known in the literature as *string-to-string transducers*. (See [4] for an example of how transducers may have many representations.) Therefore, I must define an interpretation of not only each $R_i \in \tau$ (using the formula ϕ_i) but also a σ-formula $\phi_<$ defining the linear order on the universe of the τ-structure $I(\mathfrak{A}, \alpha)$. We are particularly interested in the case where the order defined is the lexicographic order on d-tuples of $|\mathfrak{A}|$ induced by the order in $\mathfrak{A}$. This order is easily defined by a (quantifier-free) first-order formula, and we often omit it from the description of I.

When we consider interpretations that define a linear order other than the lexicographic order, we explicitly include the formula defining the order and thus the formula is $Q_d \overline{x}\overline{y}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \phi_l(\overline{y}))$, where the interpretation is of dimension d and so $|\overline{x}| = 2d$ and $|\overline{y}| = d$.

Finally, we now introduce some notation we use in the rest of the paper for various logics formed by combining particular choices of quantifiers and numerical predicates.

▶ **Definition 3.** *For a set of quantifiers $\mathfrak{Q}$ and numerical predicates $\mathfrak{N}$, we denote by $(\mathfrak{Q})[\mathfrak{N}]$ the logic constructed by extending quantifier-free first-order logic with the quantifiers in $\mathfrak{Q}$ and allowing the numerical predicates in $\mathfrak{N}$.*

We denote by FO *the set of standard first-order quantifiers: $\{\exists, \forall\}$.*

When $\mathfrak{Q}$ is just a singleton $\{Q\}$, we sometimes denote $(\mathfrak{Q})[\mathfrak{N}]$ by $(Q)[\mathfrak{N}]$ or $(Q_1)[\mathfrak{N}]$ to emphasise that it is the unary quantifier. We also write $\bar{Q}$ for the collection of all vectorizations of the quantifier Q. We use similar notation for the sets of numerical predicates. We use $\mathcal{L}((\mathfrak{Q})[\mathfrak{N}])$ to denote the languages expressible by the logic $(\mathfrak{Q})[\mathfrak{N}]$.

All the logics we consider are *substitution closed* in the sense of [10]. This means in particular that if a quantifier Q is definable in a logic $(\mathfrak{Q})[\mathfrak{N}]$, then extending the logic with the quantifier Q does not add to its expressive power. This is because we can replace occurrences of the quantifier Q by its definition, with a suitable substitution of the interpretation for the relation symbols. Hence, if Q is definable in $(\mathfrak{Q})[\mathfrak{N}]$, then $\mathcal{L}((\mathfrak{Q})[\mathfrak{N}]) = \mathcal{L}((\mathfrak{Q} \cup \{Q\})[\mathfrak{N}])$.

A remark is due on our notation for numerical predicates. All structures we consider are ordered, including those defining the quantifiers. Thus the order predicate is implicitly present in the collection of numerical predicates $\mathfrak{N}$ and is used (implicitly) to define the interpretations to a quantifier. We sometimes write $(\mathfrak{Q})[\varnothing]$ to indicate a logic in which this is the only use of the order that is allowed; by our choice of notation, the order symbol then does not appear explicitly in the syntax of the formulas.

2.4 Multiplication Quantifiers

The definition of multiplication quantifier has its origin in Barrington, Immerman, and Straubing [1, Section 5] where they were referred to as monoid quantifiers; the authors proved that the languages in NC^1 are exactly those expressible by first-order logic with quantifiers whose truth-value is determined via multiplication in a finite monoid. The notion was extended by Lautemann et al. [16] to include quantifiers for the word problem over more general algebras with a binary operation. Multiplication quantifiers over a finite monoid M can be understood as generalized quantifiers corresponding to languages recognized by M, and here we define them as such. We then see how this definition matches that of multiplication quantifiers *à la* Barrington et al. [1, 14].

Fix a monoid M, a set $B \subseteq M$, and a positive integer k. Let Σ_k denote the set $\{0,1\}^k$ which we think of as an alphabet of size 2^k, and fix a function $\gamma : \Sigma_k \to M$. We extend γ to strings in Σ_k^* inductively in the standard way: $\gamma(\epsilon) = 1_M$ and $\gamma(wa) = \gamma(w)\gamma(a)$. Note that γ is a monoid homomorphism. Together these define a language

$$L_\gamma^{M,B} = \{x \in \Sigma_k^* \mid \gamma(x) \in B\}.$$

We can now define a *multiplication quantifier*. In the following, $\mathcal{S}_L$ denotes the class of τ structures associated with a language L in the sense of Definition 2.

▶ **Definition 4.** *Let τ be a vocabulary including an order symbol $<$ and k unary relations. For a monoid M, a set $B \subseteq M$ and a function $\gamma : \{0,1\}^k \to M$, the* multiplication quantifier $\Gamma^{M,B}_{\gamma}$ *is the Lindström quantifier associated with the class of structures $\mathcal{S}_{L^{M,B}_{\gamma}}$.*

We also write $\Gamma^{M,B}_{d,\gamma}$ for the vectorization of this quantifier of dimension d. If B is a singleton $\{s\}$, then we may write $\Gamma^{M,s}_{d,\gamma}$ for short.

Recall that U_1 denotes the two-element monoid $\{0,1\}$ with standard multiplication. Then, it is easily seen that $\Gamma^{U_1,0}_{1,\gamma}$, where $\gamma : \{0,1\} \to U_1$ such that $\gamma(0) = 1$ and $\gamma(1) = 0$, is the standard existential quantifier. The universal quantifier can be defined similarly.

Another way of describing the semantics of the multiplication quantifier (which relates it directly to the form described in Barrington et al. [1]) is as follows. For a monoid M, a set $B \subseteq M$, a positive integer k and a function $\gamma : \{0,1\}^k \to M$, consider the formula

$$\Gamma^{M,B}_{d,\gamma}\overline{xy}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \varphi_k(\overline{y}))$$

where $|\overline{x}| = 2d$ and $|\overline{y}| = d$; and $I = (\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \varphi_k(\overline{y}))$ defines an interpretation of dimension d from some vocabulary τ to a vocabulary with binary relation $\sqsubset$ and k unary relations. Then, for a τ-structure $\mathfrak{A}$ and assignment α, we have that

$$\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma}\overline{xy}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \varphi_k(\overline{y}))[\alpha]$$

if, and only if, $\phi_<$ defines a linear order $\sqsubset$ on the d-tuples of $\mathfrak{A}$ and

$$\prod_{(\overline{a} \in ||\mathfrak{A}||^d)_{\sqsubset}} \gamma(\phi_1(\overline{a}), \ldots, \phi_k(\overline{a})) \in B.$$

Here, multiplication is in the monoid M and, since multiplication is not necessarily commutative, the order of d-tuples is specified to be the one given by $\sqsubset$, which is defined by $\varphi_<$.

As stated above, when $\mathfrak{A}$ is itself ordered and the order $\sqsubset$ is the lexicographic order on d-tuples, we simply omit the formula $\phi_<$.

▶ **Definition 5.** *For a monoid M, we define the following collections of quantifiers:*

$$\Gamma^M = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M,\ \gamma : \{0,1\}^k \to M,\ \text{and } d, k \geq 1\right\}$$
$$\Gamma^M_d = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M \text{ and } \gamma : \{0,1\}^k \to M\right\}$$
$$\Gamma^M_\gamma = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M,\ \text{and } d \geq 1\right\}$$
$$\Gamma^M_{d,\gamma} = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M\right\}$$

Finally, let Γ^{fin} be the collection of all multiplication quantifiers over finite monoids.

The study of the expressive power of multiplication quantifiers has generally been in relation to logics which restrict the application of multiplication quantifiers to interpretations with a lexicographic order. For this purpose, we introduce a piece of notation: we write lex-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ to mean the fragment of the logic $(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ in which all applications of quantifiers from Γ^M are to interpretations with a lexicographic order. Likewise, we write fo-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ to mean the fragment of the logic $(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ in which all applications of quantifiers from Γ^M are to interpretations in which the order is defined by a formula of (FO)[<].

From [1, Corollary 9.1], we know that NC^1 is characterized by (FO)[+, ×] equipped with finite multiplication quantifiers over lexicographic orders:

▶ **Theorem 6** ([1]). $\mathrm{NC}^1 = \mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}})[+, \times])$.

▶ Remark 7. In fact, simply adding multiplication quantifiers for some fixed finite, non-solvable monoid to (FO)[+, ×] suffices. The definition of "non-solvable monoid" is not needed for our proofs here but, for example, the *symmetric group of degree five*, denoted S_5, is a non-solvable monoid. Therefore, we know that $\mathrm{NC}^1 = \mathcal{L}(\text{lex-}(\mathrm{FO} \cup \Gamma^{S_5})[+, \times])$.

In the absence of the arithmetic predicates for addition and multiplication, the logic of multiplication quantifiers over finite monoids only allows us to define regular languages. Specifically, Barrington et al. [1, Theorem 11.1] established that the regular languages are characterized by the logic using such quantifiers with only unary interpretations.

▶ **Theorem 8** ([1]). $\textsc{Reg} = \mathcal{L}(\text{lex-}(\Gamma_1^{\mathrm{fin}})[<])$.

Later, Lautemann et al. [16, Theorem 5.1] showed that allowing interpretations of higher dimension to the quantifiers does not increase the expressive power when order is the only numerical predicate.

▶ **Theorem 9** ([16]). $\textsc{Reg} = \mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}})[<])$.

In Theorem 14 we show that this is true even in the presence of other numerical predicates and, therefore, Γ^{fin} can be replaced by Γ_1^{fin} even in Theorem 6. For our intended application, we need this technical result only in the case when the quantifier in Γ^{fin} is applied to interpretations where the order defined is lexicographic, and we give the proof in this special case. However, the result holds more generally for cases in which the quantifier in Γ^{fin} is applied to interpretations where the order is defined by a (FO)[<]-formula, and this may be of independent interest so we state the more general Theorem 19.

2.5 Typed Monoids

Our results build on the theory of *typed monoids* as developed in the work of Krebs and collaborators [2, 3, 14, 13, 6]. We recapitulate the main definitions in Appendix A for ease of reference. In particular, we need the notions of *division* of typed monoids (Definition 29), *syntactic typed monoid* (Definition 32), and *language recognition* with typed monoids (Definition 27). Here, we turn to discussing the relationship between the expressive power of logics with multiplication quantifiers and typed monoids. A formal association is defined through the definition below.

▶ **Definition 10.** *For a multiplication quantifier $Q = \Gamma_\gamma^{M,B}$ where $\gamma : \{0,1\}^k \to M$, we define the* typed quantifier monoid *of Q to be the syntactic typed monoid of the language $L_\gamma^{M,B}$.*

We also state the formal connection between the languages expressible in a logic with a collection of quantifiers and the corresponding class of typed monoids which recognizes the exact same language. For this we need the notion of the *ordered strong block product* of a pair of monoids. The definition is technical and can be found in [13] and is also reproduced in the linked full version of this paper. For a set of typed monoids T, we denote by $\mathrm{sbpc}_<(T)$ its closure under ordered strong block products.

From [13, Theorem 4.14], we then get the following relationship between logics and algebras:[2]

[2] The theorem in [13] is actually more general as it allows for more predicates than just order; however, for our purposes, order alone suffices.

▶ **Theorem 11.** *Let $\mathfrak{Q}$ be a collection of multiplication quantifiers and $\boldsymbol{Q}$ the set of typed quantifier monoids for quantifiers in $\mathfrak{Q}$. Then, $\mathcal{L}(\text{lex-}(\mathfrak{Q}_1)[<]) = \mathcal{L}(\text{sbpc}_<(\boldsymbol{Q}))$.*

Recall that the subscript in $\mathfrak{Q}_1$ means we are restricting the logic to only use the quantifiers in $\mathfrak{Q}$ on unary interpretations and the prefix "lex-" means that the quantifiers are only applied to interpretations with a lexicographic order.

3 Simplifying Multiplication Quantifiers

To use Theorem 11 to obtain an algebraic characterization of NC^1, we need to characterize this class in a logic with only unary quantifiers. Remark 7 gives us a characterization using first-order quantifiers and quantifiers in Γ^{S_5}. Our aim in this section is to show that we can eliminate the use of quantifiers of dimension higher than 1 in this logic. As a first step, we show that we can restrict ourselves to quantifiers $\Gamma_\delta^{S_5,B}$ for a *fixed* function δ.

▶ **Lemma 12.** *For every finite monoid M, there exists a function $\delta : \{0,1\}^{|M|} \to M$ such that for every $B \subseteq M$ and $\gamma : \{0,1\}^k \to M$, and dimension d, the quantifier $\Gamma_{d,\gamma}^{M,B}$ is definable in $(\Gamma_{d,\delta}^{M,B})[\varnothing]$.*

Proof. Recall that $\Gamma_{d,\gamma}^{M,B}$ is the class of structures $\mathfrak{A}$ in a vocabulary τ with one $2d$-ary ordering relation $<$ and k d-ary relations $R_1, \ldots, R_k$ such that $\gamma(w_\mathfrak{A}) \in B$ where $w_\mathfrak{A}$ is the $||\mathfrak{A}||^d$-length string associated with $\mathfrak{A}$ as in Def. 1.

Let $c = |M|$, fix an enumeration $\{m_1, \ldots, m_c\}$ of M. Let $\delta : \{0,1\}^c \to M$ be the function where $\delta(w) = m_i$ if w is the one-hot encoding of i and $\delta(w) = z$, for some arbitrary $z \in M$, otherwise (that is, if the number of occurrences of the symbol 1 in the string w is not exactly one).

For each $t \in M$, define the formula $\psi_t(y_1, \ldots, y_d)$ as follows:

$$\psi_t(y_1, \ldots, y_d) := \bigvee_{w \in \{0,1\}^k : \gamma(w) = t} \left(\bigwedge_{i \in [k] : w_i = 1} R_i(y_1, \ldots, y_d) \wedge \bigwedge_{i \in [k] : w_i = 0} \neg R_i(y_1, \ldots, y_d) \right).$$

It is easy to see that in a τ-structure $\mathfrak{A}$, we have $\mathfrak{A} \models \psi_t[a_1/y_1, \ldots, a_d/y_d]$ if, and only if, the element of $w_\mathfrak{A}$ indexed by $(a_1, \ldots, a_d)$ is mapped by γ to t. Thus, in particular, the formulas $\psi_{m_1}, \ldots, \psi_{m_c}$ define disjoint sets that partition the universe of $\mathfrak{A}$. We now claim that the quantifier $\Gamma_\gamma^{M,B}$ is defined by the formula:

$$\Gamma_\delta^{M,B} \overline{x_1 x_2} y_1 \ldots y_d (\overline{x_1} < \overline{x_2}, \psi_{m_1}(y_1, \ldots, y_d), \ldots, \psi_{m_c}(y_1, \ldots, y_d)).$$

To see this, let I denote the interpretation $(<, \psi_{m_1} \ldots, \psi_{m_c})$ so that $w_{I(\mathfrak{A})}$ is a string over $\{0,1\}^c$. By the fact that the sets defined by the formulas $\psi_{m_1}, \ldots, \psi_{m_c}$ partition $|\mathfrak{A}|$ it follows that each letter of $w_{I(\mathfrak{A})}$ is a vector in $\{0,1\}^c$ with exactly one 1. Indeed, the element of $w_{I(\mathfrak{A})}$ indexed by $(a_1, \ldots, a_d)$ is the one-hot encoding of i precisely if $\mathfrak{A} \models \psi_{m_i}[a_1/y_1, \ldots, a_d/y_d]$. Since δ takes the one-hot encoding of i to m_i, we have for any $a_1, \ldots, a_d \in |\mathfrak{A}|$

$$\begin{aligned} &\delta((w_{I(\mathfrak{A})})_{(a_1,\ldots,a_d)}) = m_i \\ \text{iff } &\mathfrak{A} \models \psi_i[a_1/y_1, \ldots, a_d/y_d] \\ \text{iff } &\gamma((w_\mathfrak{A})_{(a_1,\ldots,a_d)}) = m_i. \end{aligned}$$

Hence, $\delta(w_{I(\mathfrak{A})}) = \gamma(w_\mathfrak{A})$ and therefore $I(\mathfrak{A}) \in \Gamma_{d,\delta}^{M,B}$ if, and only if, $\mathfrak{A} \in \Gamma_{d,\gamma}^{M,B}$ as required. ◀

It then follows from Lemma 12 and the substitution property of quantifiers that the expressive power of $(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ is the same as that of $(\mathfrak{Q} \cup \Gamma^M_\delta)[\mathfrak{N}]$. Indeed, any application of a quantifier in Γ^M can be replaced by an application of a quantifier in Γ^M_δ of the same dimension. We next aim to show that an application of a quantifier $\Gamma^{M,B}_{d,\delta}$ with lexicographic order, can be replaced by d nested applications of quantifiers $\Gamma^{M,B}_{1,\delta}$

▶ **Lemma 13.** *For any collection of quantifiers $\mathfrak{Q}$ and numerical predicates $\mathfrak{N}$, any formula of* lex-$(\mathfrak{Q} \cup \Gamma^M_\delta)[\mathfrak{N}]$ *is equivalent to one of* lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$.

Proof. Again, fix an enumeration $M = \{m_1, \ldots, m_c\}$ of M and recall that $\delta : \{0,1\}^c \to M$ takes the one-hot encoding of i to m_i.

We show, by induction on d that if we have a formula

$$\Phi := \Gamma^{M,B}_{d,\delta} x_1, \ldots, x_d(\phi_1(x_1, \ldots, x_d), \ldots, \phi_c(x_1, \ldots, x_d))$$

where each formula ϕ_i is in lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$, then Φ is equivalent to a formula of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$. The result then follows by induction on the structure of the formula.

The base case when $d = 1$ is trivially true. Assume then that we have the formula Φ for $d > 1$ and let I denote the interpretation $(\phi_1(x_1, \ldots, x_d), \ldots, \phi_c(x_1, \ldots, x_d))$ of dimension d.

We claim that Φ is equivalent to the formula

$$\Phi_1 := \Gamma^{M,B}_{1,\delta} x_1(\theta_1(x_1), \ldots, \theta_c(x_1))$$

where θ_i is the formula

$$\theta_i(x_i) := \Gamma^{M,m_i}_{d-1,\delta} x_2, \ldots, x_d(\phi_1(x_1, \ldots, x_d), \ldots, \phi_c(x_1, \ldots, x_d)).$$

Thus, Φ_1 is obtained by the application of $\Gamma^{M,B}_\delta$ to an interpretation

$$I_1 := (\theta_1(x_1), \ldots, \theta_c(x_1))$$

where each formula θ_i is obtained as the application of a quantifier $\Gamma^{M,s}_{d-1,\delta}$ to an interpretation defined by formulas of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$. Thus, by the inductive hypothesis, each θ_i is equivalent to a formula of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$ and we are done.

It remains to show that Φ and Φ_1 are equivalent on any structure $\mathfrak{A}$. To see this, fix an assignment α of values in $|\mathfrak{A}|$ to the free variables of Φ. We need to show that $\mathfrak{A} \models \Phi[\alpha]$ if, and only if, $\mathfrak{A} \models \Phi_1[\alpha]$. Let n be the length of $\mathfrak{A}$ and assume without loss of generality that the elements of $|\mathfrak{A}|$ are $\{1, \ldots, n\}$ in that order.

Now, $w_{I(\mathfrak{A},\alpha)}$ denotes the string associated with the structure $I(\mathfrak{A}, \alpha)$ and note that this is a string of length n^d whose elements are indexed by d-tuples of elements of $\mathfrak{A}$ in lexicographic order. By definition, $\mathfrak{A} \models \Phi[\alpha]$ precisely if $\delta(w_{I(\mathfrak{A},\alpha)}) \in B$. We can also regard I as an interpretation of dimension $d-1$ obtained by treating the variable x_1 as a parameter. We write $w_{I(\mathfrak{A},\alpha[a/x_1])}$ for the string of length n^{d-1} that results from applying this interpretation with the assignment of a to the variable x_1. Since the ordering of d-tuples in $w_{I(\mathfrak{A},\alpha)}$ is lexicographic, we have

$$w_{I(\mathfrak{A},\alpha)} = w_{I(\mathfrak{A},\alpha[1/x_1])} \cdots w_{I(\mathfrak{A},\alpha[n/x_1])}$$

and thus

$$\delta(w_{I(\mathfrak{A},\alpha)}) = \delta(w_{I(\mathfrak{A},\alpha[1/x_1])}) \cdots \delta(w_{I(\mathfrak{A},\alpha[n/x_1])}).$$

Now, by definition of θ_i, we have $\mathfrak{A} \models \theta_i[\alpha[a/x_1]]$ if, and only if, $\delta(w_{I(\mathfrak{A},\alpha[a/x_1])}) = m_i$. Thus, for each $a \in |\mathfrak{A}|$, there is exactly one i such that $\mathfrak{A} \models \theta_i[\alpha[a/x_1]]$. Thus, $w_{I_1(\mathfrak{A},\alpha)}$ is the string of length n whose ath element is the one-hot encoding of i exactly when $\delta(w_{I(\mathfrak{A},\alpha[a/x_1])}) = m_i$. In other words, $\delta((w_{I_1(\mathfrak{A},\alpha)})_a) = \delta(w_{I(\mathfrak{A},\alpha[a/x_1])})$. Thus,

$$
\begin{aligned}
&\mathfrak{A} \models \Phi[\alpha] \\
&\text{iff } \delta(w_{I(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } \delta(w_{I(\mathfrak{A},\alpha[1/x_1])}) \cdots \delta(w_{I(\mathfrak{A},\alpha[n/x_1])}) \in B \\
&\text{iff } \delta((w_{I_1(\mathfrak{A},\alpha)})_1) \cdots \delta((w_{I_1(\mathfrak{A},\alpha)})_n) \in B \\
&\text{iff } \delta(w_{I_1(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } \mathfrak{A} \models \Phi_1[\alpha].
\end{aligned}
$$

◀

Now we are ready to state the main theorem of this section.

▶ **Theorem 14.** *For every finite monoid M, there exists a function $\delta : \{0,1\}^{|M|} \to M$ such that for any collection of quantifiers $\mathfrak{Q}$ and any set $\mathfrak{N}$ of numerical predicates, every formula of* lex-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ *is equivalent to a formula of* lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$.

Proof. Let ϕ be any formula of lex-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$. By Lemma 12 we can replace all occurrences of quantifiers in $\Gamma^{M,B}_\gamma$ by their definitions using quantifiers in $\Gamma^{M,B}_\delta$ to get a formula ϕ' of lex-$(\mathfrak{Q} \cup \Gamma^M_\delta)[\mathfrak{N}]$ equivalent to ϕ. Finally, by Lemma 13, there is a formula of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$ equivalent to ϕ'. ◀

Note that for a finite monoid M, while Γ^M and Γ^M_1 are infinite sets, $\Gamma^M_{1,\delta}$ is a finite set. Therefore, this gives us a logic characterizing NC^1 which not only uses unary quantifiers but also only has a finite number of quantifiers:

▶ **Corollary 15.** *There exists a $\delta : \{0,1\}^k \to S_5$ such that* $\mathrm{NC}^1 = \mathcal{L}(\text{lex-}(\mathrm{FO} \cup \Gamma^{S_5}_{1,\delta})[+,\times])$.

This simplifies our construction of an algebra capturing NC^1.

Another consequence of Theorem 14 is Theorem 9 ([16, Theorem 5.1]). Thus, we get a proof of Theorem 9 which is purely logical, unlike the original proof which relies on the use of finite automata. Furthermore, we also resolve a question left open in Lautemann et al. [16]:

▶ **Corollary 16.** $\mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}})[+,\times]) = \mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}}_1)[+,\times])$

4 The Algebraic Characterization

Now that we have an extension of first-order logic capturing NC^1 using only unary quantifiers, we are able to apply Theorem 11 to construct an algebra for it. We just need to obtain an equivalent logic using only the numerical predicate $<$ without introducing quantifiers of higher dimension.

To do this, we follow the construction of an algebra for TC^0. The *majority* quantifier Maj is the collection of strings over the alphabet $\{0,1\}$ in which at least half of the symbols are 1. The *square* quantifier Sq is the collection of strings over the alphabet $\{0,1\}$ in which the number of 1s is a positive square number (i.e., an element of $\mathbb{S}$). In the logics we define below, we always use these quantifiers only with unary interpretations.

The following lemma displays some known results about the expressiveness of these quantifiers:

▶ **Lemma 17.**

(i) Maj *is definable in* lex-$(\Gamma^{\text{fin}})[+,\times]$. *(cf. [1])*

(ii) *The quantifiers in* FO *are definable in* $(\text{Maj})[<]$. *([15, Theorem 3.2])*

(iii) *The numerical predicate* $+$ *is definable in* $(\text{Maj})[<]$. *([15, Theorem 4.1])*

(iv) *The numerical predicate* $\times$ *is definable in* $(\{\text{Maj},\text{Sq}\})[<]$ *and* Sq *is definable in* $(\text{Maj})[<,+,\times]$. *(cf. [18, Theorem 2.3.f] and [14, Section 2.3])*

Bringing everything together, we get the following algebraic characterization of NC^1:

▶ **Theorem 18.**

$$\text{NC}^1 = \mathcal{L}(\text{sbpc}_<(\{(\mathbb{Z},\mathbb{Z}^+,\pm 1),(\mathbb{N},\mathbb{S},\{0,1\}),(S_5,\wp(S_5),S_5)\})).$$

Proof. Let $\delta : \{0,1\}^c \to S_5$ be as defined in Lemma 12. It is easy to see that the typed quantifier monoid for Maj is $(\mathbb{Z},\mathbb{Z}^+,\pm 1)$, for Sq is $(\mathbb{N},\mathbb{S},\{0,1\})$, and for $\Gamma_{1,\delta}^{S_5,s}$ is $(S_5,\{s\},S_5)$ for any $s \in S_5$.

By Corollary 15, we have that $\text{NC}^1 = \mathcal{L}(\text{lex-}(\text{FO} \cup \Gamma_{1,\delta}^{S_5})[+,\times])$. Lemma 17 (ii)–(iv) allow us to define FO, $+$, and $\times$ in $(\{\text{Maj},\text{Sq}\})[<]$ and (i) and (iv) allow us to define Maj and Sq in lex-$(\text{FO} \cup \Gamma_{1,\delta}^{S_5})[+,\times]$. Therefore, $\text{NC}^1 = \mathcal{L}(\text{lex-}(\Gamma_{1,\delta}^{S_5} \cup \{\text{Maj},\text{Sq}\})[<])$. Moreover, we may restrict this to just quantifiers $\Gamma^{S_5,A}$ with A a singleton set, as for any $A \subseteq S_5$, the quantifier $\Gamma_{1,\delta}^{S_5,A}$ may be easily defined using Boolean combinations of quantifiers $\Gamma_{1,\delta}^{S_5,s}$ since S_5 is finite. Theorem 11 then gives us the algebraic characterization of

$$\text{NC}^1 = \mathcal{L}(\text{sbpc}_<(\{(\mathbb{Z},\mathbb{Z}^+,\pm 1),(\mathbb{N},\mathbb{S},\{0,1\})\} \cup \{(S_5,s,S_5) \mid \{s\} \in \wp(S_5)\})).$$

Because $(S_5,s,S_5) \prec (S_5,\wp(S_5),S_5)$ for all $\{s\} \in \wp(S_5)$, we lose no expressive power by replacing all elements of the form (S_5,s,S_5) with $(S_5,\wp(S_5),S_5)$. We neither gain expressive power because $\mathcal{L}((S_5,\wp(S_5),S_5)) \subseteq \text{REG} \subseteq \text{NC}^1$. Therefore, we have our final characterization:

$$\text{NC}^1 = \mathcal{L}(\text{sbpc}_<(\{(\mathbb{Z},\mathbb{Z}^+,\pm 1),(\mathbb{N},\mathbb{S},\{0,1\}),(S_5,\wp(S_5),S_5)\})).$$ ◀

5 For-Programs and Non-Lexicographic Interpretations

The proof of Lemma 13 relies crucially on the fact that we only allow lexicographic ordering of tuples in our interpretations. This is sufficient for the algebraic characterization in Section 4. However, in this section, we prove a more general version: we prove that even when interpretations are allowed to use any first-order definable ordering of tuples, we can replace multiplication quantifiers of higher arity by unary quantifiers. To do so, we rely on the work of Bojańczyk et al. [5], which characterizes first-order definable orders in terms of *d-enumerators* and *for-programs* which we review below. These are combined with techniques introduced in the proof of Theorem 14 to prove the following:

▶ **Theorem 19.** *Let* $\mathfrak{Q}$ *be any collection of quantifiers and* $\mathfrak{N}$ *any collection of numerical predicates. Then,* $\mathcal{L}(\text{fo-}(\mathfrak{Q} \cup \Gamma^{\text{fin}})[\mathfrak{N}]) = \mathcal{L}(\text{lex-}(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}])$.

As we noted in Section 2, we always assume that the collection $\mathfrak{N}$ of numerical predicates contains the order relation $<$, except in the case where it is empty. Thus, the statement of the theorem notes that in translating our formulas to use unary quantifiers, we may need to introduce the order symbol if $\mathfrak{N} = \emptyset$. From the theorem, we then get the immediate corollaries.

▶ **Corollary 20.** $\mathrm{NC}^1 = \mathcal{L}(\text{fo-}(\Gamma^{\text{fin}})[+,\times])$.

▶ **Corollary 21.** $\textsc{Reg} = \mathcal{L}(\text{fo-}(\Gamma^{\text{fin}})[<])$.

Before going into the proof, we first introduce for-programs. As the definition of d-enumerators and for-programs in general are relatively intuitive, we keep the definition brief and a more detailed treatment may be found in [5, 4]. A *first-order d-enumerator* is a for-program which takes a structure $\mathfrak{A} = (A, <, \tau^{\mathfrak{A}})$ over vocabulary τ, as input and outputs an enumeration of all the d-tuples of A. The program consists of a nesting of *for-loops* with a body:

```
for y_1 in p_1
    for y_2 in p_2
        ...
            for y_{d'} in p_{d'}
                body
```
(1)

Here the ith loop iterates the variable y_i over the domain A and p_i determines whether the iteration is in increasing order ($p_i =$ `first..last`) or decreasing order ($p_i =$ `last..first`), according to the order $<$.

The body consists of a sequence of *if-statements*

```
if θ_1(y_1,...,y_{d'}) then
    output (y_{i_1^1},...,y_{i_d^1})
if θ_2(y_1,...,y_{d'}) then
    output (y_{i_1^2},...,y_{i_d^2})
...
if θ_l(y_1,...,y_{d'}) then
    output (y_{i_1^l},...,y_{i_d^l})
```
(2)

Here $\theta_1, \ldots, \theta_l$ are $(\text{FO})[<]$-formulas over the vocabulary τ, specifying mutually exlusive conditions, so that for each assignment of values to the variables $y_1, \ldots, y_{d'}$ at most one of them is satisfied. Note that which formula is satisfied may depend on the assignment α of values to free variables other than $y_1, \ldots, y_{d'}$. We only consider programs which produce, over the course of the iteration, all tuples in A^d, with each tuple being output exactly once. For a string $\mathfrak{A}$ and assignment α, we write $P(\mathfrak{A}, \alpha)$ to mean the output of the program P when run on the string $\mathfrak{A}$ with assignment α.

From Bojańczyk et al. [5, Theorem 12], we know the following fact:

▶ **Lemma 22** ([5])**.** *For every* $(\text{FO})[<]$ *definable linear order on d-tuples, defined by a formula $\phi_<$ over a vocabulary τ of unary predicates, there exists a first-order d-enumerator P such that for every τ-structure $\mathfrak{A} = (A, <, \tau^{\mathfrak{A}})$, P on input $\mathfrak{A}$ enumerates the elements of A^d in the order defined by $\phi_<$.*

We now use this to prove Theorem 19. We first use Lemma 22 to transform a formula of $(\mathfrak{Q} \cup \Gamma^{\text{fin}})[\mathfrak{N}]$ into one in which all applications of quantifiers in Γ^{fin} use interpretations using a *generalized lexicographic order*, and then use the techniques introduced in Section 3 to transform this to a formula using only unary quantifiers.

▶ **Definition 23.** *Given a set A with a linear order $<$ on its elements and $d \in \mathbb{Z}$, a linear order $\prec$ on A^d is called a* generalized lexicographic order *if there is a sequence* $\text{dir} \in \{l, r\}^d$ *such that $\vec{a} \prec \vec{b}$ if, and only if, for the least i for which $a_i \neq b_i$, we have $a_i < b_i$ if $\text{dir}_i = l$ and $a_i > b_i$ if $\text{dir}_i = r$.*

Note that the standard lexicographic order is a generalized lexicographic order with $\text{dir} = l^d$.

Proof of Theorem 19. Let $\mathfrak{Q}$ be any collection of quantifiers and $\mathfrak{N}$ any collection of numerical predicates. Let $M = \{m_1, \ldots, m_c\}$ be a finite monoid, $B \subseteq M$, $\gamma : \{0,1\}^k \to M$, and $\phi_<$ a (FO)[<]-formula over a vocabulary τ which defines a linear ordering on d-tuples. Let $\phi_1, \ldots, \phi_k$ be fo-$(\mathfrak{Q} \cup \Gamma^{\text{fin}})[\mathfrak{N}]$-formulas over τ giving an interpretation $I = (\phi_<, \phi_1, \ldots, \phi_k)$ of dimension d. We want to show that for

$$\Phi := \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \phi_k(\overline{y}))[\alpha],$$

there is an equivalent formula in lex-$(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}]$. We first prove by induction on the nesting of multiplication quantifiers that Φ is equivalent to a formula in which all applications of quantifiers in Γ^{fin} are to interpretations in which the order defined is a generalized lexicographic order. Note that formulas (e.g., $\phi_<$) defining the order in interpretations are in (FO)[<] and, thus, by assumption do not have nested multiplication quantifiers.

For the base case, say we have a formula which uses no multiplication quantifiers; then, we are done as this is a formula of lex-$(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}]$. For the inductive step, assume that $\phi_1, \ldots, \phi_k$ are formulas of fo-$(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}]$ in which all orders are generalized lexicographical orders.

By Lemma 22, we have a first-order d-enumerator P which enumerates the elements of A^d in the order defined by $\phi_<$ on any τ-structure $\mathfrak{A}$. P has the form specified in (1) and (2). Recall that in this for-program we have l if-statements. Now, let Φ' be the formula

$$\begin{aligned}\Phi' := \Gamma^{M,B}_{d',\delta} \overline{x} y_1 \ldots y_{d'} (\xi_<(\overline{x}), &\xi_1(y_1, \ldots, y_{d'}), \ldots, \xi_c(y_1, \ldots, y_{d'}),\\ &\xi_{c+1}(y_1, \ldots, y_{d'}), \ldots, \xi_{2c}(y_1, \ldots, y_{d'}),\\ &\ldots,\\ &\xi_{(l-1)c+1}(y_1, \ldots, y_{d'}), \ldots, \xi_{lc}(y_1, \ldots, y_{d'}))\end{aligned}$$

where $\delta : \{0,1\}^{lc} \to M$ is such that for a word $w = w_1 \ldots w_{lc} \in \{0,1\}^{lc}$, $\delta(w) = m_i$ if there exists a j where $0 \leq j < l$ such that the substring $w_{jc+1} \ldots w_{(j+1)c}$ is equal to the one-hot encoding of i and all other characters in w are 0. In other words, partitioning w into l consecutive c-length substrings, we map w to m_i if one of these substrings is the one-hot encoding of i and all others are simply strings of 0s. All other elements of the domain are mapped to the identity of M. Moreover, for $0 \leq j < l$ and $1 \leq j' \leq c$, $\xi_{jc+j'}(y_1, \ldots, y_{d'})$ is the formula $\theta_j(y_1, \ldots, y_{d'}) \wedge \psi_{m_{j'}}(y_{i_1^j}, \ldots, y_{i_d^j})$ where $i_1^j, \ldots, i_d^j$ are defined as they are in the for-program P, and $\psi_{m_{j'}}$ is as defined in Lemma 12; therefore, $\xi_{jc+j'}$ is satisfied if both $\theta_j(y_1, \ldots, y_{d'})$ is and the element of M indexed by $(y_{i_1^j}, \ldots, y_{i_d^j})$ equals $m_{j'}$. Lastly, $\xi_<$ is the generalized lexicographic order given by the vector $\text{dir} \in \{l, r\}^{d'}$ with $\text{dir}_i = l$ if $p_i = \texttt{first..last}$ and $\text{dir}_i = r$ if $p_{d'} = \texttt{last..first}$. We now prove that $\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi'_1(\overline{y}), \ldots, \phi'_k(\overline{y}))[\alpha]$ iff $\mathfrak{A} \models \Phi'[\alpha]$.

Let 1_M denote the identity of M and $f_{\mathfrak{A},\alpha} : A^d \cup \{\epsilon\} \to M$ be the function defined by

$$f_{\mathfrak{A},\alpha}(a_1, \ldots, a_d) = \gamma({\phi'_1}^{\mathfrak{A},\alpha}[a_1, \ldots, a_d], \ldots, {\phi'_k}^{\mathfrak{A},\alpha}[a_1, \ldots, a_d])$$

for $(a_1, \ldots, a_d) \in A^d$ and $f_{\mathfrak{A},\alpha}(\epsilon) = 1_M$. and let $g_{\mathfrak{A},\alpha} : A^{d'} \to A^d \cup \{\epsilon\}$ be the function defined by the body of the for-program P where if no output is produced, $g_{\mathfrak{A},\alpha}(a_1, \ldots, a_{d'}) = 0^d$; thus,

$$g_{\mathfrak{A},\alpha}(a_1, \ldots, a_{d'}) = \begin{cases} (a_{i_1^j}, \ldots, a_{i_d^j}) & \text{if } \exists j \text{ such that } \mathfrak{A} \models \theta_j(a_1, \ldots, a_{d'}) \\ \epsilon & \text{o.w.} \end{cases}$$

Let $f^*_{\mathfrak{A},\alpha} : (A^d)^* \to M$ and $g^*_{\mathfrak{A},\alpha} : (A^{d'})^* \to (A^d)^*$ be the monoid homomorphisms induced by the functions $f_{\mathfrak{A},\alpha}$ and $g_{\mathfrak{A},\alpha}$, respectively, in the natural way.

Let $\mathfrak{A}$ be an arbitrary τ-structure and α an arbitrary variable assignment with assignments for at least all free variables in each ϕ'_i with the exception of $\overline{y}_i$. By definition of multiplication quantifiers, we get that $\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi'_1(\overline{y}), \ldots, \phi'_k(\overline{y}))[\alpha]$ iff $\gamma(w_{I(\mathfrak{A},\alpha)}) \in B$.

By construction of $f_{\mathfrak{A},\alpha}$ and because P provides the order by which $\Gamma^{M,B}_{d,\gamma}$ is evaluated, it is easy to see that $\gamma(w_{I(\mathfrak{A},\alpha)}) \in B$ iff $f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha)) \in B$.

Let $t_1, \ldots, t_{|A|^{d'}}$ denote the ordering of d'-tuples of A as defined by $\xi_<$. Because the output of P is an enumeration of A^d in the ordering defined by $\phi_<$, and $g_{\mathfrak{A},\alpha}$ outputs the empty string ϵ during an iteration of its for-loops only when P outputs nothing at that iteration, it follows that

$$f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) = f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha))$$

and, thus, $f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) \in B$ iff $f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha)) \in B$.

Let $I' = (\xi_<, \xi_1, \ldots, \xi_{lc})$. We now want to show that

$$\delta(w_{I'(\mathfrak{A},\alpha)}) \in B \text{ iff } f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) \in B.$$

To do so, we prove that $\delta((w_{I'(\mathfrak{A},\alpha)})_a) = f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_a))$ for every $0 \leq a \leq |A|^{d'}$. Let a be arbitrary. Then, $u = (w_{I(\mathfrak{A},\alpha)})_a$ is either 0^{lc} or a one-hot encoding of some $b \in [lc]$. If $u = 0^{lc}$, then $\delta(u) = 1_M$ and no θ_j is satisfied. Thus, $g^*_{\mathfrak{A},\alpha}(t_a) = \epsilon$ and $f^*_{\mathfrak{A},\alpha}(\epsilon) = 1_M$ by construction. If u is a one-hot encoding of some $b \in [lc]$, then some θ_j is satisfied and,

$$\delta(u) = m = \gamma({\phi'_1}^{\mathfrak{A},\alpha}[a_{i_1^j}, \ldots, a_{i_d^j}], \ldots, {\phi'_k}^{\mathfrak{A},\alpha}[a_{i_1^j}, \ldots, a_{i_d^j}]) = f_{\mathfrak{A},\alpha}(a_{i_1^j}, \ldots, a_{i_d^j})$$

by definition of δ and the construction of each ξ_i. By the construction of $g_{\mathfrak{A},\alpha}$ and $f_{\mathfrak{A},\alpha}$, we also get immediately that $g^*_{\mathfrak{A},\alpha}(t_a) = (a_{i_1^j}, \ldots, a_{i_d^j})$ and, thus, $f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_a)) = m$. Thus, $\delta(w_{I'(\mathfrak{A},\alpha)}) \in B$ iff $f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{|A|^{d'}})) \in B$.

Then, by definition, we have that $\mathfrak{A} \models \Phi'[\alpha]$ iff $\delta(w_{I'(\mathfrak{A},\alpha)}) \in B$.

Therefore, all together, we have that

$$\begin{aligned}
&\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi'_1(\overline{y}), \ldots, \phi'_k(\overline{y}))[\alpha] \\
&\text{iff } \gamma(w_{I(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha)) \in B \\
&\text{iff } f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) \in B \\
&\text{iff } \delta(w_{I'(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } \mathfrak{A} \models \Phi''[\alpha].
\end{aligned}$$

This concludes the inductive step. Therefore, for any formula Φ, we have an equivalent formula Φ' which uses only uses multiplication quantifiers using a generalized lexicographic order.

To see that we can construct a formula equivalent to Φ' using only unary quantifiers, we use the technique used in the proof of Lemma 13 with a minor modification to account for generalized lexicographic order. In the proof of Lemma 13, a quantifier $\Gamma^{M,B}_{d,\gamma}$ of dimension d is replaced by a sequence of d nested unary quantifiers over the same monoid M. To account for the order $\xi_<$, which defines a generalized lexicographic order with $\text{dir} \in \{l, r, \}^d$, we define the ith quantifier in the sequence as before if $\text{dir}_i = l$ and replace the monoid $(M, \cdot)$ with $(M, \cdot^R)$ where $m \cdot^R m' := m' \cdot m$ for all $m, m' \in M$ if $\text{dir}_i = r$. The proof follows exactly along the lines of the proof of Lemma 13.

All together, we have an equivalent formula to Φ', and, thus, Φ, which only uses unary quantifiers, completing the proof. ◀

6 Conclusion

In this work, we constructed a class of typed monoids exactly recognizing NC^1. To do so, we proved results regarding the expressive power of logics with quantifiers defined over finite monoids. Specifically, we established that the expressive power is not changed by restricting the dimension of the interpretations on which the quantifiers act, regardless of which numerical predicates are available.

Therefore, we were able to provide a logic characterizing NC^1 which only uses unary quantifiers and use this logic to construct an algebraic characterization of NC^1. This result marks the second circuit complexity class to be characterized in a such a way, with the first being TC^0 [14], and provides a decomposition theorem in the style of Krohn-Rhodes for NC^1.

An interesting future direction would be to construct similar algebraic characterization of other complexity classes beyond NC^1. It seems this would require the development of new algebraic tools. In particular, the block product is a key tool used to characterize first-order quantification and more generally quantification over interpretations of dimension one. To extend the work to other complexity classes, it would be worthwhile investigating other product constructions that might similarly relate to quantification on interpretations of higher dimension as well as higher-order quantifiers.

Moreover, we extended the work of Barrington et al. [1] and Lautemann et al. [16] by studying the expressive power of multiplication quantifiers when applied to interpretations using other than the standard lexicographic order. We showed that over strings the expressive power of logics equipped with finite multiplication quantifiers is not changed by loosening the restriction to permitting any first-order definable linear order. A natural next step would be to investigate the expressive power when other linear orders are permitted. For example, by extending the Domination Lemma of Bojańczyk et al. [5] to monadic second-order logic (MSO[<]), one would be able to show that permitting the application of multiplication quantifiers to interpretations with an MSO[<]-definable order does not change the expressive power. This would result in the interesting consequence that

$$\textsc{Reg} = \mathcal{L}(\text{lex-}(\Gamma_1^{\text{fin}})[<]) = \mathcal{L}((\Gamma^{\text{fin}})[<]).$$

References

1 David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

2 Christoph Behle, Andreas Krebs, and Mark Mercer. Linear circuits, two-variable logic and weakly blocked monoids. In *International Symposium on Mathematical Foundations of Computer Science*, pages 147–158. Springer, 2007. doi:10.1007/978-3-540-74456-6_15.

3 Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid. Typed monoids–An Eilenberg-like theorem for non regular languages. In *Algebraic Informatics: 4th International Conference, CAI 2011, Linz, Austria, June 21-24, 2011. Proceedings 4*, pages 97–114. Springer, 2011. doi:10.1007/978-3-642-21493-6_6.

4 Mikolaj Bojańczyk. Transducers of polynomial growth. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–27, 2022.

5 Mikolaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-string interpretations with polynomial-size output. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, page 106. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.106.

6 Antonio Cano, Jesus Cantero, and Ana Martínez-Pastor. A positive extension of Eilenberg's variety theorem for non-regular languages. *Applicable Algebra in Engineering, Communication and Computing*, 32(5):553–573, 2021. doi:10.1007/S00200-020-00414-2.

7 Stephen A Cook. Characterizations of Pushdown Machines in Terms of Time-Bounded Computers. *Journal of the ACM (JACM)*, 18(1):4–18, 1971. doi:10.1145/321623.321625.

8 Anuj Dawar and Aidan T Evans. Characterizing NC1 with typed monoids. *arXiv preprint arXiv:2508.11019*, 2025. doi:10.48550/arXiv.2508.11019.

9 H-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

10 Heinz-Dieter Ebbinghaus. Extended logics: The general framework. In Jon Barwise and Solomon Feferman, editors, *Model-Theoretic Logics*, pages 25–76. Springer-Verlag, New York, 1985.

11 Samuel Eilenberg. *Automata, Languages, and Machines (Vol. B)*. Academic Press, 1976.

12 Fred C Hennie. One-tape, off-line turing machine computations. *Information and Control*, 8(6):553–578, 1965. doi:10.1016/S0019-9958(65)90399-2.

13 Andreas Krebs. *Typed Semigroups, Majority Logic, and Threshold Circuits*. PhD thesis, Tübingen, Univ., Diss., 2008, 2008.

14 Andreas Krebs, Klaus-Jörn Lange, and Stephanie Reifferscheid. Characterizing TC0 in terms of infinite groups. *Theory of Computing Systems*, 40(4):303–325, 2007. doi:10.1007/S00224-006-1310-2.

15 Klaus-Jörn Lange. Some results on majority quantifiers over words. In *Proceedings. 19th IEEE Annual Conference on Computational Complexity, 2004.*, pages 123–129. IEEE, 2004. doi:10.1109/CCC.2004.1313817.

16 Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. *Journal of Computer and System Sciences*, 62(4):629–652, 2001. doi:10.1006/JCSS.2000.1742.

17 John L Rhodes. *Applications of Automata Theory and Algebra: Via the Mathematical Theory of Complexity to Biology, Physics, Psychology, Philosophy, and Games*. World Scientific, 2010.

18 Nicole Schweikardt. *On the Expressive Power of First-order Logic with Built in Predicates*. Logos-Verlag, 2002.

19 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

20 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer Science & Business Media, 1999.

A Typed Monoids

In this section, we review the definitions and results from [3, 13] on typed monoids, their relationship to languages and corresponding characterizations of complexity classes.

A typed monoid is a monoid equipped with a collection of *types*, which form a Boolean algebra, and a set of *units*. We only deal with concrete Boolean algebras, given as collections of subsets of a fixed universe.

▶ **Definition 24** (Boolean Algebra). *A* Boolean algebra *over a set S is a set $B \subseteq \wp(S)$ such that $\varnothing, S \in B$ and B is closed under union, intersection, and complementation. If B is finite, we call it a* finite *Boolean algebra.*

We call $\varnothing$ and S the trivial elements *(or in some contexts, the* trivial types*) of B.*

A homomorphism between Boolean algebras is defined as standard. That is, if B_1 and B_2 are Boolean algebras over sets S and T, respectively, then we call $h : B_1 \to B_2$ a *homomorphism* if $h(\varnothing) = \varnothing$, $h(S) = T$, and for all $s_1, s_2 \in B_1$, $h(s_1 \cap s_2) = h(s_1) \cap h(s_2)$, $h(s_1 \cup s_2) = h(s_1) \cup h(s_2)$, and $h(s^C) = (h(s))^C$. Now we are ready to define typed monoids.

▶ **Definition 25** (Typed Monoid). *Let M be a monoid, G a Boolean algebra over M, and E a finite subset of M. We call the tuple $T = (M, G, E)$ a* typed monoid over M *and the elements of G* types *and the elements of E* units*. We call M the* base monoid *of T. If M is a group, then we may also call T a* typed group*.*

When $G = \{\varnothing, A, M - A, M\}$ for some $A \subseteq M$, we abbreviate T as (M, A, E), i.e., the Boolean algebra is signified by an element, or elements, which generates it – in this case, A.

We say that a typed monoid (M, G, E) is finite if M is.

We also need a notion of morphism between typed monoids.

▶ **Definition 26.** *A* typed monoid homomorphism h *from* (S, G, E) *to* (T, H, F) *is a triple* (h_1, h_2, h_3) *where* $h_1 : S \to T$ *is a monoid homomorphism,* $h_2 : G \to H$ *is a homomorphism of Boolean algebras, and* $h_3 : E \to F$ *is a mapping of sets such that the following conditions hold:*

(i) *For all* $A \in G$, $h_1(A) = h_2(A) \cap h_1(S)$.

(ii) *For all* $e \in E$, $h_1(e) = h_3(e)$.

Note that h_3 is redundant in the definition as it is completely determined by h_1. We retain it as part of the definition for consistency with [3, 13].

To motivate the definitions, recall that a language $L \subseteq \Sigma^*$ is recognized by a monoid M if there is a homomorphism $h : \Sigma^* \to M$ and a set $B \subseteq M$ such that $L = h^{-1}(B)$. When the monoid M is infinite, the languages recognized form a rather rich collection and we aim to restrict this in two ways. First, B cannot be an arbitrary set but must be an element of the algebra of types. Secondly, the homomorphism h must map the letters in Σ to units of the typed monoid. Formally, we have the following definition.

▶ **Definition 27.** *A typed monoid* $T = (M, G, E)$ recognizes *a language* $L \subseteq \Sigma^*$ *if there exists a typed monoid homomorphism from* (Σ^*, L, Σ) *to* T. *We let* $\mathcal{L}(T)$ *denote the set of languages recognized by* T.

When the base monoid of a typed monoid is finite, we recover the classical definition of a recognition. Hence, the languages recognized by finite typed monoids are necessarily regular.

▶ **Proposition 28.** *If* T *is a finite typed monoid, then* $\mathcal{L}(T) \subseteq$ Reg.

We can now state the definitions of the key relationships between typed monoids.

▶ **Definition 29.** *Let* (S, G, E) *and* (T, H, F) *be typed monoids.*

- *A typed monoid homomorphism* $h = (h_1, h_2, h_3) : (S, G, E) \to (T, H, F)$ *is* injective *(*surjective*, or* bijective*) if all of* h_1, h_2, *and* h_3 *are.*
- (S, G, E) *is a* typed submonoid *(or, simply, "submonoid" when context is obvious) of* (T, H, F), *denoted* $(S, G, E) \leq (T, H, F)$, *if there exists an injective typed monoid homomorphism* $h : (S, G, E) \to (T, H, F)$.
- (S, G, E) *divides* (T, H, F), *denoted* $(S, G, E) \preceq (T, H, F)$, *if there exists a surjective typed monoid homomorphism from a submonoid of* (T, H, F) *to* (S, G, E).

These have the expected properties.

▶ **Proposition 30** ([3])**.** *Let* T_1, T_2, *and* T_3 *be typed monoids.*

- *Typed monoid homomorphisms are closed under composition.*
- *Division is transitive: if* $T_1 \preceq T_2$ *and* $T_2 \preceq T_3$, *then* $T_1 \preceq T_3$.
- *If* $T_1 \preceq T_2$, *then* $\mathcal{L}(T_1) \subseteq \mathcal{L}(T_2)$.

We can formulate the notion of the *syntactic typed monoid* of a language L as an extension of the syntactic monoid of L with a minimal collection of types and units necessary.

▶ **Definition 31.** *Let $T = (M, G, E)$ be a typed monoid. A congruence $\sim$ over M is a* typed congruence over T *if for every $A \in G$ and $s_1, s_2 \in M$, if $s_1 \sim s_2$ and $s_1 \in A$, then $s_2 \in A$.*

For a typed congruence $\sim$ over T, let

$$A/{\sim} = \{[x]_\sim \mid x \in A\} \text{ where } A \subseteq M$$
$$G/{\sim} = \{A/{\sim} \mid A \in G\}$$
$$E/{\sim} = \{[x]_\sim \mid x \in E\}.$$

Then, $T/{\sim} := (M/{\sim}, G/{\sim}, E/{\sim})$ is the typed quotient monoid of T by $\sim$.

Let $\sim_T$ denote the typed congruence on T such that for $s_1, s_2 \in S$, $s_1 \sim_T s_2$ if, and only if, for all $x, y \in S$ and $A \in G$, $xs_1y \in A$ if, and only if, $xs_2y \in A$. We then refer to the quotient monoid $T/{\sim_T}$ as the minimal reduced monoid of T.

Recall that $\sim_L$ is the syntactic congruence of L, defined in Section 2.2.

▶ **Definition 32.** *For a language $L \subseteq \Sigma^*$, the* syntactic typed monoid of L, *denoted* $\mathrm{syn}(L)$, *is the typed monoid $(\Sigma^*, L, \Sigma)/{\sim_L}$.*

We also get the *canonical typed monoid homomorphism*, $\eta_L : (\Sigma^*, L, \Sigma) \to \mathrm{syn}(L)$ induced by the syntactic homomorphism of L.

It also turns out that we can give a purely structural characterization of those typed monoids that are syntactic monoids.

▶ **Proposition 33** ([13])**.** *A typed monoid is the syntactic monoid of a language if, and only if, it is reduced, generated by its units, and has four or two types.*

In case it has just two types, then it only recognizes the empty language or the language of all strings.