A Family of Effective Methods for Decompiling Canonical Acceptors, Instantiated for Languages of Dot-Depth One and Tier-Based Extensions

Dakotah Lambert
Department of Mathematics and Computer Science
Lake Forest College
dakotahlambert@acm.org

Abstract

Many kinds of logical systems have been employed in constructing formal languages to model phonological phenomena. A common theme among them is that the systems compile into finite automata. Two questions naturally arise. Can a given phenomenon be described with another logical system? And, if so, what is that description?

To the first question, algebraic techniques are well established through deep connections with logic and automata. To the second, the situation is less clear. Translations from automata are established for first-order and monadic second-order logics under precedence, but these may not translate easily to the simpler systems we often use. Translations for simple cases of restricted propositional logic (strictly local or strictly piecewise languages) are established, but insufficient to describe attested phenomena.

The present work establishes a general way to handle many systems in between. Specifically, we show how to translate between certain kinds of algebraic varieties $\mathbf{V}$ (systems defined by universally satisfied identities) and associated logical systems, then use decomposition to handle classes of the form $\mathbf{V} * \mathbf{D}$, where the notion of "symbol" is replaced by "k-block". With this, we handle several (unrestricted) propositional logics, facilitating logical description of natural language.

1 Introduction

Logical formalisms and their correspondence to finite automata provide a rich and well-grounded base for descriptions of natural language and linguistic theories (see Rogers, 1998; Hulden, 2009; Heinz, 2018, 2025). There are many tools that convert logical descriptions into finite automata, including MONA by Henriksen et al. (1995) focused on monadic second-order logic, Foma by Hulden (2009) focused on first-order logic, or the Language Toolkit by Lambert (2024) focused on quantifier-free systems of propositional logic. Compiling logical expressions to automata is solved. The present work addresses the inverse: decompiling (or "factoring") automata into human-readable logical expressions.

Algebraic techniques provide a universal approach to deciding whether a given automaton belongs to a given class, whether it can be written in a particular logical form (Lambert, 2026). The Language Toolkit provides these tools, but, in general, these techniques do not provide the logical description. The Büchi–Elgot–Trakhtenbrot theorem (Büchi, 1960; Elgot, 1961; Трахтенброт, 1961) provides a means to convert from automata back into monadic second-order logic. McNaughton and Papert (1971) show how to convert appropriate automata back into first-order logic with precedence (less-than). Below, for all but the weakest logics, less is available.

A language is locally testable in the strict sense (strictly local) if it is defined by a collection of forbidden substrings. A language is piecewise testable in the strict sense (strictly piecewise) if it is defined by a collection of forbidden subsequences. In either case, any presence of a forbidden factor immediately rules out the word. Rogers and Lambert (2019) provide mechanisms to factor automata into conjunctive systems of constraints forbidding substrings and/or subsequences, thereby handling these restricted propositional systems. Janoušek and Plachý (2024) provide an alternative approach for factoring strictly local systems. The vast space between here and first-order seems largely unexplored.

The present work addresses this gap. We provide a general approach, grounded in algebraic techniques, to convert from automata to a wide range of propositional logics. A Python implementation is provided as an appendix. Any algebraic class with a (parameterized) finite free object (see §2.3)

and interpretable elements can be handled. By decomposing systems of the form $\mathbf{V} * \mathbf{D}$ (see §4), we generalize those results to systems wherein the notion of "symbol" is replaced by "k-block". A k-block is a contiguous substring of length k, the same kind of structure considered in an n-gram model.

This paper presents these methods and illustrates them with respect to the analysis of phonological patterns in Navajo, Karanga Shona, Tsuut'ina, Luganda, and stress patterns in the StressTyp2 database of Goedemans et al. (2015). These examples are chosen because they illustrate increasingly complex classes between strictly local or strictly piecewise and first-order logic. In particular, after establishing the basic method for content languages (Navajo, §3), this method is extended to the locally testable class (Karanga Shona, §4), the tier-based locally testable class (Tsuut'ina, §6), the locally threshold testable class (Luganda, §7), and the piecewise testable and dot-depth one classes (stress patterns, §8). Importantly, the core structure of the algorithm is identical throughout.

2 Background

2.1 Machines and Structures

Throughout this work, determinism will be assumed, so the word "deterministic" will be omitted outside of definitions. A **deterministic finite semiautomaton** is defined by a finite alphabet (Σ), a finite set (Q) of states, and a transition function that maps a letter–state pair to a state. Let $\delta_x(q)$ denote the application of this transition function to the input consisting of the letter $x \in \Sigma$ at the state $q \in Q$. This transition function is extended to words by $\delta^*_\lambda(q) = q$ and $\delta^*_{uv} = \delta^*_v \circ \delta^*_u$, where λ represents the unique word of length zero. A **deterministic finite-state acceptor** (DFA) is a deterministic finite semiautomaton augmented with an initial state ($q_0 \in Q$) and a set ($Q_f \subseteq Q$) of accepting states. We write these components as a five-tuple $A = \langle \Sigma, Q, \delta, q_0, Q_f \rangle$. A DFA defines a language L such that a word w is in L if, and only if, $\delta^*_w(q_0) \in Q_f$.

A state $q \in Q$ is **reachable** if there exists some word $w \in \Sigma^*$ such that $\delta^*_w(q_0) = q$). Two states q and q' in Q are **distinguishable** if there exists some word $w \in \Sigma^*$ such that $\delta^*_w(q) \in Q_f$ but $\delta^*_w(q') \notin Q_f$ or vice versa. An automaton is said to be in canonical form, or simply **canonical**, if all states are reachable and pairwise distinguishable.

A **deterministic finite-state transducer** (DFT) is a deterministic finite-state acceptor that associates computation with output. There is an output alphabet (Γ), and Γ^* is the set of words composed of letters in Γ. A DFT has a universal prefix ($\pi \in \Gamma^*$), a transition-output function (γ) where $\gamma_x(q) \in \Gamma^*$ is the output emitted upon encountering the letter $x \in \Sigma$ in state $q \in Q$, and $\sigma(q) \in \Gamma^*$ is the output emitted upon stopping in the state $q \in Q_f$. The γ function extends to words much like δ: $\gamma^*_\lambda(q) = \lambda$ and $\gamma^*_{xw}(q) = \gamma_x(q)\gamma^*_w(\delta_x(q))$. If w is a word such that $\delta^*_w(q_0) \in Q_f$, then the output associated with w is $\pi\gamma^*_w(q_0)\sigma(\delta^*_w(q_0))$, otherwise there is no such output.

A **semigroup** is a set alongside an associative operation. Function composition is associative; the set of δ^*_w for $w \in \Sigma^+$ is therefore a semigroup, where Σ^+ is the set of nonempty words composed of characters in Σ. This is called the **transition semigroup** of the semiautomaton. As these are functions from a finite domain to a finite codomain, there are only finitely many such functions, so every semiautomaton is associated with a finite semigroup. The transition semigroup of a DFA in some sense describes the structure of its associated language: if $\delta^*_u = \delta^*_v$, this means that there is a sense in which u and v have equivalent behavior. Namely, for any context $a__b$, it holds that $\delta^*_{aub} = \delta^*_b \circ \delta^*_u \circ \delta^*_a = \delta^*_b \circ \delta^*_v \circ \delta^*_a = \delta^*_{avb}$, so aub is accepted if, and only if, avb is accepted. Let $u \sim v$ denote this equivalence. Equivalence is maintained by concatenation; it is easy to verify that if $u \sim v$ and $x \sim y$, then $ux \sim uy \sim vy \sim vx$. The **syntactic semigroup** of a language L is the transition semigroup of a canonical automaton that represents L.

Let suff_k be the function that retrieves the k-suffix of a word; that is, $\mathrm{suff}_k(\lambda) = \mathrm{suff}_0(w) = \lambda$ for $w \in \Sigma^*$ and $\mathrm{suff}_k(ua) = \mathrm{suff}_{k-1}(u)a$ for $u \in \Sigma^*$ and $a \in \Sigma$. A DFT is **input strictly k-local** if $\delta^*_w(q_0) = \delta^*_{\mathrm{suff}_{k-1}(w)}(q_0)$ for all $w \in \Sigma^*$ (Chandlee et al., 2014). If $q \in Q$ is an arbitrary reachable state, then $q = \delta^*_v(q_0)$ for some $v \in \Sigma^*$. If the length of w is at least $k-1$, then $\mathrm{suff}_{k-1}(w) = \mathrm{suff}_{k-1}(vw)$ no matter what v is. From this we derive the following for any input strictly k-local transducer:

$$\begin{aligned}\delta^*_w(q) = \delta^*_{vw}(q_0) &= \delta^*_{\mathrm{suff}_{k-1}(vw)}(q_0) \\ &= \delta^*_{\mathrm{suff}_{k-1}(w)}(q_0) = \delta^*_w(q_0)\end{aligned}$$

That is, for sufficiently long strings, their corresponding transition functions are constant. This in

turn means that, so long as every state is reachable, we have not just that $\delta^*_w(q_0) = \delta^*_{\text{suff}_{k-1}(w)}(q_0)$, but in general $\delta^*_w = \delta^*_{\text{suff}_{k-1}(w)}$. In other words, for any string w in Σ^* it holds that $w \sim \text{suff}_{k-1}(w)$.

Lambert and Heinz (2023) point out that this corresponds to an algebraic property called "definiteness". A semigroup is **k-definite** if for any elements u and x_1, x_2, ..., x_k, it holds that $ux_1x_2 \dots x_k = x_1x_2 \dots x_k$. An input strictly k-local transducer has a transition semigroup that is $(k-1)$-definite. Let a and $b_1 \dots b_{k-1}$ be strings in Σ^+, and let $u = \delta^*_a$ and $x_i = \delta^*_{b_i}$ for $1 \leq i < k$. Because each b_i contributes at least one letter, we have the following.

$$\begin{aligned} ux_1 \dots x_{k-1} &= \delta^*_{ab_1 \dots b_{k-1}} \\ &= \delta^*_{\text{suff}_{k-1}(ab_1 \dots b_{k-1})} = \delta^*_{\text{suff}_{k-1}(b_1 \dots b_{k-1})} \\ &= \delta^*_{b_1 \dots b_{k-1}} = x_1 \dots x_{k-1} \end{aligned}$$

In terms of universal algebra, a **variety** is a collection of structures that satisfy a system of universally satisfied identities, where specific elements cannot be referenced, only variables. The k-definite semigroups satisfy $ux_1x_2 \dots x_k = x_1x_2 \dots x_k$ where the u and each x_i range over all elements; we write the corresponding variety as $\mathbf{D}_k = [\![ux_1x_2 \dots x_k = x_1x_2 \dots x_k]\!]$. This says that $\mathbf{D}_k$ is the set of semigroups that satisfy this identity. If the system were built from multiple identities, they would be written separated by semicolons.

2.2 Propositional Logics

We work with a classical truth-functional propositional logic, in which formulae are interpreted as being either true or false (contrasting with modal logics). Formulae are built from a collection of atoms and standard logical connectives of $\wedge$ (and), $\vee$ (or), and $\neg$ (not); quantifiers are not available.

Let Σ be a finite alphabet and $\mathcal{W}$ be (a model of) a word. We say $\mathcal{W} \vDash \phi$ (read "$\mathcal{W}$ satisfies ϕ") if the interpretation of the formula ϕ over $\mathcal{W}$ is true. We now define our atoms. Define $\text{content}(\mathcal{W})$ to be the set of symbols that occur in $\mathcal{W}$. Each symbol $x \in \Sigma$ is an atom, where $\mathcal{W} \vDash x$ means $x \in \text{content}(\mathcal{W})$. That is, $\mathcal{W} = uxv$ for some strings u and v. The atom $x_1x_2 \dots x_n$ (where each $x_i \in \Sigma$) represents occurrence as a **substring**: $\mathcal{W} \vDash x_1x_2 \dots x_n$ means $\mathcal{W} = ux_1x_2 \dots x_nv$ for some strings u and v. The atom $x_1 \,..\, x_2 \,..\, \cdots \,..\, x_n$ (where each $x_i \in \Sigma$) represents occurrence as a **subsequence**: $\mathcal{W} \vDash x_1 \,..\, x_2 \,..\, \cdots \,..\, x_n$ means that there exists some sequence $u_1, \dots, u_n$ of words of strictly increasing length such that $\mathcal{W} = u_ix_iv_i$ (for some v_i) for each x_i. This essentially means that the x_i occur in $\mathcal{W}$ in order, but not necessarily adjacently. This generalizes to subsequences of substrings: $\mathcal{W} \vDash x_{1.1} \dots x_{1.n} \,..\cdots..\, x_{m.1} \dots x_{m.n}$ means that the substrings $x_{i.1} \dots x_{i.n}$ occur in order, but not necessarily adjacently, with potential overlap. For example, we have that "ash" $\vDash$ as .. sh. Call this a **generalized subsequence**.

Let ϕ and ψ be logical formulae. The basic logical connectives are conjunction ($\phi \wedge \psi$, read "ϕ and ψ"), disjunction ($\phi \vee \psi$, read "ϕ or ψ"), and negation ($\neg\phi$, read "not ϕ"). We have that $\mathcal{W} \vDash \phi \wedge \psi$ if both $\mathcal{W} \vDash \phi$ and $\mathcal{W} \vDash \psi$, that $\mathcal{W} \vDash \phi \vee \psi$ if $\mathcal{W} \vDash \phi$ or $\mathcal{W} \vDash \psi$, or both, and that $\mathcal{W} \vDash \neg\phi$ if $\mathcal{W} \nvDash \phi$. A formula is in **disjunctive normal form** if it is a disjunction of one or more conjunctions of literals, where a literal is an atom or its negation. Our factorization systems provide formulae in disjunctive normal form.

2.3 Free Objects and Blockifiers

A **free** object in a class is the most generic form of a structure, where no equalities hold other than those implied by the definition of the class. For instance, Σ^+ is the free semigroup generated by Σ, as for any two sequences $u_1 \dots u_n$ and $v_1 \dots v_m$ of letters drawn from Σ, the resulting words in Σ^+ are equal if, and only if, $n = m$ and $u_i = v_i$ for $1 \leq i \leq n$. A free k-definite semiautomaton has a distinct state for each string of length up to k. The transition function is defined such that $\delta_a(w) = \text{suff}_k(wa)$.

We define a particular input $(k+1)$-strictly local transducer over this structure that we shall call a **$(k+1)$-blockifier**. The input alphabet shall be Σ, and the output alphabet shall consist of blocks of length $k+1$, potentially including boundaries:

$$\mathbb{B}_{k+1}(\Sigma) = \Sigma^{k+1} \cup (\rtimes \cdot \Sigma^k) \cup (\Sigma^k \cdot \ltimes) \cup (\rtimes \cdot \Sigma^{\leq k-1} \cdot \ltimes)$$

The states are those of the free k-definite semiautomaton: let w denote both a string and its corresponding state. Let the initial state q_0 be λ, indicating that no input has yet been seen, let the set of accepting states be $Q_f = Q$, and let the universal prefix π be λ. Consider a string u of length up to k and a letter $x \in \Sigma$. We have $\delta_x(u) = \text{suff}_k(ux)$, while the output functions depend on the length of

u, written $|u|$:

$$\gamma_x(u) = \begin{cases} \boxed{ux} & \text{if } |u| = k \\ \boxed{\rtimes ux} & \text{if } |u| = k-1 \\ \lambda & \text{otherwise} \end{cases}$$

$$\sigma(u) = \begin{cases} \boxed{u\ltimes} & \text{if } |u| = k \\ \boxed{\rtimes u\ltimes} & \text{otherwise} \end{cases}$$

This guarantees that every output symbol is either a fully bounded word whose length, including boundaries, is at most $k+1$, or it is an optionally bounded block of size exactly $k+1$.

The k-blockifier maps an input word w to the sequence of k-blocks in w in order from left to right, with overlap. For instance, the 3-blockifier applied to the word "abacaba" would emit $\boxed{\rtimes ab}\boxed{aba}\boxed{bac}\boxed{aca}\boxed{cab}\boxed{aba}\boxed{ba\ltimes}$. Throughout, let $B_k(w)$ be this function, mapping w through the k-blockifier to this output.

The blockifier is a transducer built with the structure of a free k-definite semigroup. Other free objects will be instrumental to our factorization methods.

3 Content Languages

Heinz (2010), citing Sapir and Hoijer (1967), analyzes a symmetric sibilant harmony system in Navajo, wherein the anteriority of sibilants is influenced by that of the rightmost sibilant. The surface constraint that arises from such a system is that a [+anterior] sibilant like 's' cannot occur in the same word as a [−anterior] sibilant like 'ʃ'. The acceptability of a word with respect to this constraint depends solely on its content. This means that, for this constraint, the order in which symbols occur is irrelevant ($uv \sim vu$ for all u and v), as is their quantity ($uu \sim u$ for all u). We call such a language a **content-language**. They can be described in a propositional logic whose only atoms are of the form $x \in \Sigma$. The variety $[\![uv = vu; uu = u]\!]$ is called $\mathbf{J}_1$ (see Almeida, 1995).

Eilenberg's Theorem (1976) associates with each variety $\mathbf{V}$ of semigroups a corresponding family $\mathcal{V}$ of languages: a language over Σ belongs to $\Sigma\mathcal{V}$ if, and only if, its syntactic semigroup belongs to $\mathbf{V}$. This family is thus also called a variety. For a language in $\Sigma\mathcal{J}_1$, by imposing an order on the symbols of the alphabet, words have a normal form reached by repeatedly using the $uv \sim vu$ rule to sort the symbols, then repeatedly using the $uu \sim u$ rule to deduplicate them. Equivalence is maintained throughout; we have $w \sim \text{content}(w)$. For a finite alphabet Σ there are only finitely many such forms. This fact indicates that the free object in $\Sigma\mathcal{J}_1$ is finite: there is one element per subset of Σ.

So a language L over Σ is in $\Sigma\mathcal{J}_1$ if, and only if, its syntactic semigroup is in $\mathbf{J}_1$. This does not, however, tell us what formula describes the language. But because the free object for $\Sigma\mathcal{J}_1$ is finite, we can extract this information as follows. Let $A = \langle \Sigma, Q, \delta, q_0, Q_F \rangle$ be the canonical acceptor for L. We maintain a stack of pairs of the form $\langle q, F \rangle$ where $q \in Q$ is a state in A and $F \subseteq \Sigma$ is an element of our free object. (This corresponds to a depth-first search. Replacing the stack with a queue performs a breadth-first search instead. Both are equally viable.) We also maintain a visited set of such pairs, a set of acceptable symbol sets and a set of rejected symbol sets. (In general, these are acceptable and rejected elements of the free object.) Let $R_\lambda(F) = F$ and $R_x(F) = F \cup \{x\}$, representing the transition function in a semiautomaton derived from our free object, and let $\alpha = \varnothing$, representing what would be the initial state in an acceptor so derived.

1. Begin with an empty visited set and a stack consisting of the single item $\langle q_0, \alpha \rangle$.
2. While the stack is nonempty:
   (a) Pop $\langle q, F \rangle$ from the stack and add it to the visited set.
   (b) For each alphabet symbol $x \in \Sigma$, push $\langle \delta_x(q), R_x(F) \rangle$ if it is unvisited.
   (c) If $q \in Q_f$, accept F.
   (d) If $q \notin Q_f$, reject F.

This process builds up the necessary portion of the free $\Sigma\mathcal{J}_1$ object, essentially unwrapping the canonical acceptor onto it. At the end of this process, the collection of acceptable symbol sets is interpretable as a formula in disjunctive normal form defining L.

The translation is as follows. Let $S = \{x_1, \ldots, x_m\}$ be an acceptable symbol set and let $\{y_1, \ldots, y_n\} = \Sigma - S$ be the symbols not in that set. Words that arrive at this element are those where each x_i appears and no y_i appears:

$$(x_1 \wedge \cdots \wedge x_m \wedge \neg y_1 \wedge \cdots \wedge \neg y_n)$$

Words are accepted if, and only if, they arrive at any such element, so the logical formula is the disjunction of the formulae derived from all acceptable symbol sets. In general, this approach produces

a logical formula for any variety with a finite free object whose elements are interpretable like this.

4 Blockifiers and Local Testability

A language is **locally k-testable** if the acceptability of a word w depends only on its set of substrings of length k, including word boundaries (McNaughton and Papert, 1971). Whether a shorter word of length less than $k-2$ is included is entirely arbitrary. This means that the acceptability of $w \in \Sigma^*$ is determined by $\mathrm{content}(B_k(w))$. A language is **locally testable** if, and only if, it is locally k-testable for some finite k.

So a language L is locally testable if, and only if, there is some language in $\mathbb{B}_k(\Sigma)\mathcal{J}_1$ that accepts $B_k(w)$ for all those w in L and rejects $B_k(w)$ for all those w not in L. That is, there is some DFA M whose transition semigroup is in $\mathbf{J}_1$ such that L is the language of $M \circ B_k$. In algebraic terms, the operator corresponding to this composition is the wreath product. The variety $\mathbf{V} * \mathbf{D}_{k-1}$ corresponds to the class of languages that can be defined by a composition of B_k into a DFA whose transition semigroup is in $\mathbf{V}$ (Straubing, 1985).

One example of a linguistically relevant locally testable pattern is the surface pattern of tone on non-assertive verb stems in Karanga Shona, a Bantu language described by Odden (1984) and analyzed by Jardine (2020) when defining the class of melody-local languages. Per Jardine (2020), there are two basic kinds of roots which pattern as follows, where superscript $+$ indicates one or more occurrences of the symbol it is attached to:

- H-toned: $\{\mathrm{H}, \mathrm{HL}, \mathrm{HLH}, \mathrm{HHLH}\} \cup \mathrm{HHHL}^+\mathrm{H}$
- L-toned: $\{\mathrm{L}, \mathrm{LH}, \mathrm{LHL}\} \cup \mathrm{LHHL}^+$

The thirteen-state canonical DFA recognizing the union of these two sets is shown in Figure 1, with the thirteenth, rejecting, sink state omitted. Each circle is a state, those with doubled outlines are accepting, and the transition function is indicated by overlaying the letter on an arrow from the source state to its destination. We assert that this is locally testable, and will verify this via factorization.

If we have an automaton $A = \langle \Sigma, Q, \delta, q_0, Q_f \rangle$ whose language we know to be locally testable, then we know that there must be some $k \geq 1$ for which this language is in $\mathbf{J}_1 * \mathbf{D}_{k-1}$. By fixing a correct value for k, we can iteratively build up the free object in $\mathbb{B}_k(\Sigma)\mathcal{J}_1$ much like we did in the previous

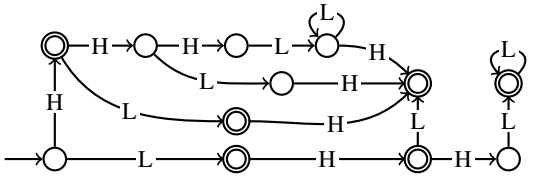

Figure 1: Karanga Shona non-assertive verb stems.

section, except that its input is not drawn from the acceptor A but from the output of the k-blockifier B_k as it is traced alongside A. We extend our stack to track not pairs but triples, consisting of an A-state, a B_k-state, and the corresponding set of output symbols (k-blocks). We still maintain a visited set, an acceptable set, and a rejected set. Still let $R_\lambda(F) = F$ and $R_x(F) = F \cup \{x\}$ otherwise, and $\alpha = \varnothing$ as we are working over the same free object structure.

1. Begin with an empty visited set and a stack consisting of the single item $\langle q_0, \lambda, \alpha \rangle$.
2. While the stack is nonempty:
   (a) Pop $\langle q, w, F \rangle$; mark it visited.
   (b) Push $\langle \delta_x(q), \mathrm{suff}_{k-1}(wx), R_{\gamma_x(w)}(F) \rangle$ for each symbol x, if not visited.
   (c) If $q \in Q_f$, accept $R_{\sigma(w)}(F)$.
   (d) If $q \notin Q_f$, reject $R_{\sigma(w)}(F)$.

This process builds up the necessary portion of the free $\mathbf{J}_1$ machine generated by k-blocks. Each acceptable set generated will have one prefix and one suffix (which may be merged as a fully anchored word) and zero or more factors without boundaries. The corresponding formula is the same disjunctive normal form as for content-languages, except that the atoms derived from symbols (which represent k-blocks) are interpreted as the substrings that they represent.

By this process, fixing $k = 4$, the following fifteen factor sets are emitted as valid for Karanga Shona non-assertive verb stems. Thirteen factor sets capture a single word apiece; these are $B_4(w)$ for w selected from H, L, HL, LH, HLH, LHL, HHLH, LHHL, LHHLL, HHHLH, HHHLLH, LHHLLL, or HHHLLLH. The remaining two have components that can be overlaid to produce arbitrarily long words, these are $\{\rtimes$LHH, LHHL, HHLL, HLLL, LLLL, LLL$\ltimes\}$, representing words of the form $\mathrm{LHHLLLL}^+$, and $\{\rtimes$HHH, HHHL, HHLL, HLLL, LLLL, LLLH, LLH$\ltimes\}$, representing words of the form $\mathrm{HHHLLLL}^+\mathrm{H}$.

5 Rejection or Approximation

If a language is not locally k-testable, then there must be two strings, u and v, which have the same set of k-blocks, but one is accepted while the other is rejected. This situation is detectable: if during the above process a factor set F is marked as acceptable when it has already been marked as rejected or vice versa, this indicates that the language is not locally k-testable and that some pair of wrongly indistinguishable u and v share factor set F. There are a few options at this point. On the one hand, one could choose to terminate the procedure and simply declare that the language is not locally k-testable and no factorization can exist. On the other hand, one could choose to continue as normal: by caring only about acceptable elements, we would say that every word w that shares a factor set with an acceptable word w' will be accepted, even if it should not be. Words whose factor set is never accepted will continue to be rejected. This is the smallest locally k-testable language that is a superset of the target language, a tight approximation. Flipping this, we can get the largest locally k-testable language that is a subset of the target language by rejecting all rejected elements. We can insist on correct factorization, or we can give a tight upper- or lower-bound.

This sensitivity allows factorization of arbitrary locally testable languages, even if we do not know the parameter k in advance. One way is to simply begin with the assumption that $k = 1$. If factorization fails because the language is not locally 1-testable, then try again with $k = 2$, and so on, increasing k by one each time. The result will be a factorization with the minimal possible value for k. This is the method that was used for Karanga Shona non-assertive verb stems.

Another approach is to use a theorem of Straubing (1985) to find a k that is guaranteed to work. Given two semigroup elements x and y, we say that $x \leq_{\mathcal{R}} y$ if, and only if, $x = y$ or there is some s such that $x = ys$. Similarly, we say that $x \leq_{\mathcal{L}} y$ if, and only if, $x = y$ or there is some s such that $x = sy$. The strict versions of these inequalities are defined such that $x <_{\mathcal{R}} y$ means $x \leq_{\mathcal{R}} y$ but $y \not\leq_{\mathcal{R}} x$ and similarly for $<_{\mathcal{L}}$. A chain under an inequality, $<$, is a sequence $x_1 < x_2 < \cdots < x_n$. Per Straubing (1985, Theorem 5.3), if a language is in $\mathbf{V} * \mathbf{D}_k$ for some (potentially unknown) k and c is the length of the longest chain under $<_{\mathcal{R}}$ or under $<_{\mathcal{L}}$, whichever is shorter, then that language is in $\mathbf{V} * \mathbf{D}_c$. As there are only finitely many elements in the syntactic semigroup of any regular language, this is effectively calculable. However, this does not necessarily yield the minimal value! Using this approach would have resulted in factoring with $k = 6$ instead of the minimal $k = 4$.

6 Nonsalient Symbols and Tier Analysis

Often in natural language patterns, we encounter dependencies that are nonlocal when considering a word as a whole, but become local when looking only at a particular class of symbols. The asymmetric sibilant harmony of Tsuut'ina reported by Cook (1978a,b) and analyzed by Heinz (2010) is one such case, where a [−anterior] sibilant may not follow a [+anterior] sibilant at any distance, but the reverse situation is allowed: /si-tʃiz-aʔ/ "my duck" surfaces as **ʃítʃídz**à, for example. Unlike the symmetric harmony of Navajo, this is not a content-language. Heinz (2010) analyzes this pattern in two different ways. A precedence based constraint could forbid the subsequence s…ʃ (but allow ʃ…s), Or a tier-based analysis could ignore symbols other than sibilants and forbid the substring sʃ on the result. The latter approach is expanded by Heinz et al. (2011) and explored by Lambert (2023).

We can effect such analyses by taking advantage of a key insight of Lambert (2023): nonsalient symbols (here, non-sibilants) are those that never change state. They are the symbols that label self-loops on every state. To analyze such a pattern, first remove the nonsalient symbols, then do the factorization on the projected result, and state that the resulting constraints apply on a tier. Here, if we assume a limited alphabet such as $\Sigma = \{\text{a}, \text{s}, \text{ʃ}\}$ we would use $k = 2$ and the tier $\text{T} = \{\text{s}, \text{ʃ}\}$. Running the analysis, the permissible factor sets on the tier T turn out to be $\{\rtimes\ltimes\}$, $\{\rtimes\text{s}, \text{s}\ltimes\}$, $\{\rtimes\text{ʃ}, \text{ʃ}\ltimes\}$, $\{\rtimes\text{ʃ}, \text{ʃs}, \text{s}\ltimes\}$, $\{\rtimes\text{ʃ}, \text{ʃʃ}, \text{ʃ}\ltimes\}$, $\{\rtimes\text{s}, \text{ss}, \text{s}\ltimes\}$, $\{\rtimes\text{ʃ}, \text{ʃʃ}, \text{ʃs}, \text{s}\ltimes\}$, $\{\rtimes\text{ʃ}, \text{ʃs}, \text{ss}, \text{s}\ltimes\}$, and $\{\rtimes\text{ʃ}, \text{ʃʃ}, \text{ʃs}, \text{ss}, \text{s}\ltimes\}$. In no case is sʃ permitted.

7 Multiplicity and Threshold Testability

The set of symbols in a word w is $\text{content}(w)$. Define $\text{content}_t(w)$ to be the *multiset* of symbols in w, counted up to some threshold t, beyond which further occurrences cannot be distinguished. If $uv\ vu$ and $uu^t\ u^t$, then we have $w \sim \text{content}_t(w)$: repeatedly apply the first to sort symbols, then repeatedly apply the second to deduplicate as much as possible. These identities define the variety $\mathbf{Acom}_t = [\![uv = vu; uu^t = u^t]\!]$. There is no finite free ob-

ject for $\mathbf{Acom}$ in general: such an object must be able to make all possible distinctions in terms of count but there are infinitely many natural numbers to distinguish. However, the family of parameterized classes satisfies $\mathbf{Acom}_1 \subseteq \mathbf{Acom}_2 \subseteq \cdots$. Extraction of $\mathbf{Acom}_t$ proceeds like that of $\mathbf{J}_1$ as seen in §3 Let $R_x(F)$ insert x into F, increasing its occurrence-count by one if it is not already t and let α be the empty multiset. If the parameter t is not known in advance, it can be derived from the procedure of §5: begin with $t = 1$, and if this fails, try again with increasingly large values for t. As each increase yields a superclass, if the system is $\mathbf{Acom}$ at all, this will eventually succeed.

The **locally threshold-t k-testable** languages are the generalization from symbols to k-blocks here, $\mathbf{Acom}_t * \mathbf{D}_{k-1}$, where the acceptability of a word is determined by its multiset of k-blocks, counted with multiplicity up to some threshold t, beyond which additional occurrences are not distinguished (see McNaughton and Papert, 1971 or Beauquier and Pin, 1989). A language is **locally threshold testable** if, and only if, it is locally threshold-t k-testable for some finite k and t. The $t = 1$ case is local testability; $k = 1$ is $\mathcal{Acom}$.

By Straubing's Theorem (1985), k is bounded above by the maximal length of a chain under $<_{\mathcal{R}}$ or $<_{\mathcal{L}}$, so we can use whichever is shorter. We can then either find t analytically, by building all products of k semigroup elements and for each such product p finding the smallest value t_p where $p^{t_p} = pp^{t_p}$, then selecting the largest of the resulting t_p. Or we can run the blockified factorization procedure of §4, beginning from $t = 1$ and increasing this value, as above, if inconsistencies arise.

One locally threshold testable pattern that arises in natural language is culminativity of stress, the constraint that at most one syllable with primary stress occurs (Hyman, 2014). This constraint belongs to many classes, in fact. For instance, it is tier-based locally testable (in the strict sense): if p represents primary stress and u represents lack of stress, then the 2-block pp is forbidden on the tier $\mathrm{T} = \{p\}$. It belongs to $\mathcal{Acom}$, as the multiset of symbols suffices to determine whether a word is acceptable, but Straubing's Theorem (1985) gives an upper bound of $k = 2$. Using $k = 2$ and $t = 1$, factorization fails; the pattern is not locally 2-testable. Using $k = 2$ and $t = 2$, the system returns the following collection of fourteen valid 2-multisets: $\{\rtimes\ltimes\}$, $\{\rtimes u, u\ltimes\}$, $\{\rtimes p, p\ltimes\}$, $\{\rtimes u, uu, u\ltimes\}$, $\{\rtimes u, up, p\ltimes\}$, $\{\rtimes p, pu, u\ltimes\}$, $\{\rtimes u, uu, uu, u\ltimes\}$, $\{\rtimes u, uu, up, p\ltimes\}$, $\{\rtimes u, up, pu, u\ltimes\}$, $\{\rtimes p, pu, uu, u\ltimes\}$, $\{\rtimes u, up, pu, uu, u\ltimes\}$, $\{\rtimes u, uu, up, pu, uu, u\ltimes\}$. The block pp is never permissible, nor is any configuration in which pu or up appears twice. Finally, if a word starts with p it cannot contain up and if a word ends with p it cannot contain pu.

Another locally threshold testable constraint is unbounded high-tone plateauing, discussed by Jardine (2020) in relation to Luganda, in which tones may be low or high but surface forms only have a single span of high tones. That is, they may not contain a substring of the form $\mathrm{HL}^+\mathrm{H}$. Straubing's Theorem (1985) gives an upper bound of $k = 4$, but the system can be factored with $k = t = 2$. The resulting set of 34 valid 2-multisets serve to forbid three constructs. Words cannot start with H and contain LH. Words cannot end with H and contain HL. And words cannot contain HL twice.

8 Subsequences and Dot-Depth One

The piecewise testable languages were studied by Simon (1975). A language is **piecewise j-testable** if the acceptability of a word depends only on the set of j-subsequences in the word. A language is **piecewise testable** if it is piecewise j-testable for some finite j. The algebraic variety that corresponds to this class is typically known as $\mathbf{J}$. Like with $\mathbf{Acom}$, there is not a finite free object for $\mathbf{J}$, but $\mathbf{J}_1 \subsetneq \mathbf{J}_2 \subsetneq \mathbf{J}_3 \subsetneq \cdots$ and each $\mathbf{J}_j$ does have a finite free object. The free object for $\mathbf{J}_j$ has elements that correspond to possible sets of subsequences of length up to j, where if u is in the set and x is a subsequence of u, then x too is in the set. For such a system, the initial element is $\alpha = \{\lambda\}$ and $R_x(F) = F \cup \{\mathrm{suff}_j(sx) : s \in F\}$.

The dot-depth hierarchy was introduced by Cohen and Brzozowski (1971). A language has **dot-depth one** when the acceptability of a word depends on its set of j-subsequences of k-blocks, for some finite j and k. In other words, the class of languages of dot-depth one is $\mathbf{J} * \mathbf{D}$ (Knast, 1983; Straubing, 1985). The restriction to $j = 1$ yields the locally k-testable languages, and the restriction to $k = 1$ yields the piecewise j-testable languages.

Rogers and Lambert (2019) analyze and factor the patterns with associated automata in the StressTyp2 database of Goedemans et al. (2015). A result of this is that all but six of the properly subregular patterns factor into a combination of constraints that are locally testable in the strict sense, the com-

plement of such a constraint (which remains locally testable), or piecewise testable in the strict sense. The remaining six patterns have additional locally testable constraints that could not be extracted automatically but were instead supplied manually. However, this means that all of these properly subregular patterns are the combination of a locally testable component and a piecewise testable component; they have dot-depth one. The factorization methodology we present can fully factor all of these patterns, obviating the need for manual analysis even in these complex systems.

9 Conclusions

We presented a family of effective methods for extracting grammars of classes of the form $\mathbf{V} * \mathbf{D}$ from a canonical DFA. The block size k may be fixed or known in advance, or an overestimate can be derived by taking the length of the longest chain under $<_{\mathcal{R}}$ or $<_{\mathcal{L}}$, whichever is shorter. Alternatively, you can begin by assuming $k = 1$ and try again if necessary with increasing large k.

One selling point of the Language Toolkit of Lambert (2024) is that an analyst can describe a language in whatever terms are most convenient, then determine whether the language so-described is representable in various logical systems. We add to this now the capability to determine what that alternative representation actually is, at least for anything that is a content-language ($\mathbf{J}_1$), a content_t-language ($\mathbf{Acom}$), a piecewise j-testable language ($\mathbf{J}_j$), or the result of composing an input strictly k-local transducer into such a system, namely a locally k-testable language ($\mathbf{J}_1 * \mathbf{D}$), a locally threshold-$t$ k-testable language ($\mathbf{Acom} * \mathbf{D}$), or a language with dot-depth one ($\mathbf{J} * \mathbf{D}$), or the (single) tier-based extension of any of these.

In general, given a variety $\mathbf{V}$ with a finite free object, you need to define three things: you need a function R that maps an element of the free object and a symbol to another element of the free object, you need an initial element α, and you need a logical interpretation of the elements of the free object. From there, the procedure is as follows. Maintain a stack (or queue) of triples consisting of the current state in the DFA, the current state in a free input strictly k-local transducer, and the current element of the free object. Maintain also a set of visited triples, a set of acceptable elements of the free object, and a set of its rejected elements.

1. Begin with an empty visited set and a stack consisting of the single item $\langle q_0, \lambda, \alpha \rangle$.
2. While the stack is nonempty:
   (a) Pop $\langle q, w, F \rangle$; mark it visited.
   (b) Push $\langle \delta_x(q), \text{suff}_{k-1}(wx), R_{\gamma_x(w)}(F) \rangle$ for each symbol x, if not visited.
   (c) If $q \in Q_f$, accept $R_{\sigma(w)}(F)$.
   (d) If $q \notin Q_f$, reject $R_{\sigma(w)}(F)$.
3. Replace symbol k-blocks by their meaning.

If your desired variety $\mathbf{V}$ does not have a finite free object, but it can be parameterized into a family of subvarieties $\mathbf{V}_1 \subsetneq \mathbf{V}_2 \subsetneq \mathbf{V}_3 \subsetneq \cdots$, where each $\mathbf{V}_i$ does have such an object, then all is not lost. Like with the block size, it is possible to begin with the assumption that the parameter is 1, then try again with increasingly large values if that fails. If ever an element is marked as both acceptable and rejected, then either the language is not a member of $\mathbf{V} * \mathbf{D}$ or some of the parameters are too small.

For locally testable factorization, $\alpha = \varnothing$ and $R_z(F) = F \cup \{z\}$. For locally threshold-t testable systems, these parameters are the same, except that $\varnothing$ and F are not sets but t-multisets. For dot-depth one, $\alpha = \{\lambda\}$ and $R_z(F) = F \cup \{\text{suff}_j(sz) : s \in F\}$. In the end, the process will return valid sets of k-blocks, valid t-multisets of k-blocks, or valid sets of j-subsequences of k-blocks, respectively.

We demonstrated the utility of these techniques by factoring several phenomena from a range of phonological domains, including long-distance dependencies that arise in harmony, stress, and tone. We improved upon the results of Rogers and Lambert (2019), who factored the patterns in the StressTyp2 database but needed to supply additional constraints manually for six properly subregular patterns. Each of those six has dot-depth at most one, so the techniques demonstrated herein can be applied to avoid this burden. Similarly, we improved over the work of Janoušek and Plachý (2024), who presented a different method of factoring patterns that are locally testable in the strict sense. We maintain the ability to produce tight superset (or subset) approximations.

By stripping nonsalient symbols, performing the analysis, then restating the result in terms of a projective tier, this family of factorization methods extends trivially to the (single) tier-based extensions of factorable classes. Reasonable methods for handling multiple tiers remain an open problem.

Other future directions include searching for better parameter-finding methods and performing more analyses on more patterns in more languages.

References

Jorge Almeida. 1995. *Finite Semigroups and Universal Algebra*, volume 3 of *Series in Algebra*. World Scientific, Singapore.

Danièle Beauquier and Jean-Éric Pin. 1989. Factors of words. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca, editors, *Automata, Languages and Programming: 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 63–79. Springer Berlin / Heidelberg.

Julius Richard Büchi. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6(1–6):66–92.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

Rina S. Cohen and Janusz Antoni Brzozowski. 1971. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16.

Eung-Do Cook. 1978a. Palatalizations and related rules in Sarcee. In Eung-Do Cook and Jonathan Kaye, editors, *Linguistic Studies of Native Canada*, pages 19–35. University of British Columbia Press, Vancouver, Canada.

Eung-Do Cook. 1978b. The synchronic and diachronic status of Sarcee γ^y. *International Journal of American Linguistics*, 44(3):192–196.

Samuel Eilenberg. 1976. *Automata, Languages, and Machines*, volume B. Academic Press, New York, New York.

Calvin C. Elgot. 1961. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51.

R. W. N. Goedemans, Jeffrey Heinz, and Harry van der Hulst. 2015. StressTyp2.

Jeffrey Heinz. 2010. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. In Larry Hyman and Frank Plank, editors, *Phonological Typology*, volume 23 of *Phonetics and Phonology*, chapter 5, pages 126–195. Mouton de Gruyter.

Jeffrey Heinz. 2025. The logical structure of phonological generalizations. *Phonological Studies*, 28:69–80.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Short Papers*, volume 2, pages 58–64, Portland, Oregon. Association for Computational Linguistics.

Jesper Gulmann Henriksen, Jakob Jensen, Michael Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. 1995. MONA: Monadic second-order logic in practice. Technical Report RS-95-21, BRICS, Department of Computer Science, University of Aarhus, Aarhus, Denmark.

Mans Hulden. 2009. *Finite-State Machine Construction Methods and Algorithms for Phonology and Morphology*. Ph.D. thesis, The University of Arizona.

Larry M. Hyman. 2014. Do all languages have word accent? In Harry van der Hulst, editor, *Word stress: Theoretical and typological issues*, pages 56–82. Cambridge University Press.

Jan Janoušek and Štěpán Plachý. 2024. Shortest characteristic factors of a deterministic finite automaton and computing its positive position run by pattern set matching. In *SOFSEM 2024: Theory and Practice of Computer Science*, volume 14519 of *Lecture Notes in Computer Science*, pages 326–339. Springer.

Adam Jardine. 2020. Melody learning and long-distance phonotactics in tone. *Natural Language & Linguistic Theory*, 38:1145–1195.

Robert Knast. 1983. A semigroup characterization of dot-depth one languages. *RAIRO – Informatique théorique*, 17(4):321–330.

Dakotah Lambert. 2023. Relativized adjacency. *Journal of Logic, Language and Information*, 32(4):707–731.

Dakotah Lambert. 2024. System description: A theorem-prover for subregular systems: The Language Toolkit and its interpreter, plebby. In *Functional and Logic Programming: 17th Annual Symposium, FLOPS 2024*, volume 14659 of *Lecture Notes in Computer Science*, pages 311–328, Kumamoto, Japan. Springer, Singapore.

Dakotah Lambert. 2026. Multitier phonotactics with logic and algebra. *Phonology*, 43:e9 1–31.

Dakotah Lambert and Jeffrey Heinz. 2023. An algebraic characterization of total input strictly local functions. In *Proceedings of the Society for Computation in Linguistics*, volume 6, pages 25–34, Amherst, Massachusetts.

Robert McNaughton and Seymour Aubrey Papert. 1971. *Counter-Free Automata*. MIT Press.

David Odden. 1984. Stem tone assignment in Shona. In George N. Clements and John Goldsmith, editors, *Autosegmental Studies in Bantu Tone*. Foris Publications, Dordrecht, The Netherlands.

James Rogers. 1998. *A Descriptive Approach to Language-Theoretic Complexity*. (Monograph.) Studies in Logic, Language, and Information. CSLI Publications.

James Rogers and Dakotah Lambert. 2019. Extracting Subregular constraints from Regular stringsets. *Journal of Language Modelling*, 7(2):143–176.

Edward Sapir and Harry Hoijer. 1967. *The Phonology and Morphology of the Navaho Language*, volume 50 of *Linguistics*. University of California Press, Berkeley, California.

Imre Simon. 1975. Piecewise testable events. In Helmut Brakhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer-Verlag, Berlin.

Howard Straubing. 1985. Finite semigroup varieties of the form $V * D$. *Journal of Pure and Applied Algebra*, 36:53–94.

Борис Авраамович Трахтенброт. 1961. Конечные Автоматы и Логика Одноместных Предикатов. *Доклады Академии наук СССР*, 140(2):326–329.

A Python Implementation

```
def lt():
    def R(F, sym):
        if sym is None: return F
        return F | frozenset([sym])
    return R, frozenset()
def ltt(t):
    def R(F, sym):
        if sym is None: return F
        d = dict(F)
        d[sym] = min(t, d.get(sym, 0) + 1)
        return frozenset(d.items())
    return R, frozenset()
def dd1(j):
    def R(F, sym):
        if sym is None: return F
        return ( F
               | frozenset([(sym,)])
               | frozenset((s + (sym,))[-j:] for s in F))
    return R, frozenset([tuple()])

def blockifier(k, state, sym=None):
    if sym is None:
        if len(state) < k: return '^' + state + '$'
        if k == 0: return None
        return state + '$'
    if len(state) == k: return state + sym
    if len(state) == k - 1: return '^' + state + sym
    return None

def factorize(k, R, alpha, aut, *args, approximate=False):
    if k < 1: raise ValueError('k must be at least one')
    stack = [(aut['q0'], '', alpha)]
    visited, accepted, rejected = set(), set(), set()
    while stack:
        q, islq, F = stack.pop()
        visited.add((q, islq, F))
        nexts = set()
        for letter,qn in aut['delta'][q].items():
            nexts.add((aut['delta'][q][letter],
                       (islq + letter)[-(k - 1):][:k - 1],
                       R(F, blockifier(k - 1, islq, letter))))
        m = R(F, blockifier(k - 1, islq))
        if q in aut['Qf']:
            if m in rejected and not approximate: return None
            accepted.add(m)
        else:
            if m in accepted and not approximate: return None
            rejected.add(m)
        stack.extend(nexts - visited)
    return accepted
```

Here are some examples. Output is omitted, but each runs nearly instantly.

```
karanga_shona = {
    'delta': {
        0 : {'H':  0, 'L':  0}, # rejecting sink
        1 : {'H':  2, 'L':  9},
        2 : {'H':  3, 'L':  8},
        3 : {'H':  4, 'L':  7},
        4 : {'H':  0, 'L':  5},
        5 : {'H':  6, 'L':  5},
        6 : {'H':  0, 'L':  0},
        7 : {'H':  6, 'L':  0},
        8 : {'H':  6, 'L':  0},
        9 : {'H': 10, 'L':  0},
        10: {'H': 11, 'L':  6},
        11: {'H':  0, 'L': 12},
        12: {'H':  0, 'L': 12},
    },
    'q0': 1,
    'Qf': set([2, 6, 8, 9, 10, 12])
}
print(*factorize(4, *lt(), karanga_shona), sep='\n')
```

```
culminativity = {
    'delta': {
        0 : {'p':  0, 'u':  0}, # rejecting sink
        1 : {'p':  2, 'u':  1},
        2 : {'p':  0, 'u':  2},
    },
    'q0': 1,
    'Qf': set([1,2])
}
print(*factorize(2, *ltt(2), culminativity), sep='\n')
```

```
murik = { # from StressTyp2
    'delta': { # alphabet: w0.s0=l, w0.s2=L, w1.s2=H
        0: { 'l':0, 'L':0, 'H':0 }, # rejecting sink
        1: { 'l':3, 'L':2, 'H':2 },
        2: { 'l':2, 'L':0, 'H':0 },
        3: { 'l':3, 'L':0, 'H':2 },
    },
    'q0': 1,
    'Qf': set([2])
}
print(*factorize(1, *dd1(2), murik), sep='\n')
```