Finite-State Methods for Linguistics: On Deriving and Checking Constraints

A Dissertation Presented

by

Dakotah Jay Lambert

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Linguistics

Stony Brook University

May 2022

Stony Brook University

The Graduate School

Dakotah Jay Lambert

We, the dissertation committee for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation.

Jeffrey Heinz — Dissertation Advisor
Professor, Department of Linguistics and Institute for Advanced Computational Science

Thomas Graf — Chairperson of Defense
Associate Professor, Department of Linguistics

Jordan Kodner
Assistant Professor, Department of Linguistics

Owen Rambow
Professor, Department of Linguistics and Institute for Advanced Computational Science

James Rogers
Professor Emeritus, Department of Computer Science, Earlham College

This dissertation is accepted by the Graduate School

Eric Wetheimer
Dean of the Graduate School

Abstract of the Dissertation

Finite-State Methods for Linguistics: On Deriving and Checking Constraints

by

Dakotah Jay Lambert

Doctor of Philosophy

in

Linguistics

Stony Brook University

2022

This dissertation continues a long tradition of research that began with Kleene (1956) and Chomsky (1959) regarding the description of linguistic patterns in a way that guarantees a constant memory bound on processing. The regular languages are all and only those which can be represented in this way, but smaller subclasses of patterns provide more useful properties with respect to learnability (Gold, 1967) or the machinery required to decide inclusion (McNaughton and Papert, 1971).

I expand upon the prior literature in four ways. I characterize in various ways some classes of patterns, including the complements of strictly local languages as well as those classes defined by constraints over tiers: the tier-based strictly local class of Heinz et al. (2011) and newly introduced extensions thereof. I discuss restricted first-order logics with respect to the piecewise-local subregular hierarchy. I introduce strongly directed hypergraphs to unify tree acceptors and relations with their string-based counterparts. Finally, as a cautionary tale, I show that the semiring analysis of Lothaire (2005) which unifies acceptors and relations is as powerful as a Turing machine.

Table of Contents

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Review of the Literature 3

1.2 Outline of the Dissertation 8

2 Formal Languages and String Acceptors 11

2.1 Formal Language Theory 11

2.2 Finite Model Theory 12

2.3 Graphs and Finite-State Automata 17

2.4 Notation 19

3 Classifying and Factoring Tier-Based Extensions of the Subregular Hierarchy 21

3.1 Model Theoretic Descriptions 23

3.2 Language-Theoretic Characterizations 26

3.2.1 Strict locality 26

3.2.2 Complements 29

3.2.3 Local testability 31

3.2.4 Threshold testability 32

3.2.5 Piecewise relativizations 33

3.3 Automata 34

3.3.1 Characterizations 35

3.3.2 Constructions 37

3.4 Closure Properties 40

3.4.1 Products 40

3.4.2 Complements of automata 42

3.4.3 Some non-closures 43

3.5 Algebra 45

3.5.1 Strictly local stringsets and their complements 48

3.5.2 Locally testable stringsets 48

3.5.3 Locally threshold testable stringsets 49

3.6 Conclusions 50

4 Learning Tier-Based Strictly Local Languages 53

4.1 Preliminaries 54

4.1.1 (Tier-Based) Strict Locality 54

4.1.2 Our Learning Problem 56

4.1.3 String Extension Learning 57

4.2 Deciding Salience 58

4.3 The Substructures 60

4.4 Pointwise String Extension Learning 62

4.5 A Worked Example of the Final Simplified Approach 64

4.6 Non-Strict Locality 65

4.7 Conclusions 66

5 **Monoid Varieties and a Subregular Spiral** **69**

5.1 Background 70

5.1.1 Green's Relations 73

5.2 Tier-Based Classes of Languages 74

5.3 Specific Classes of Formal Languages 76

5.3.1 Piecewise Testable 78

5.3.2 Semilattice Languages 78

5.3.3 Locally Testable 79

5.3.4 Generalized Definite 79

5.3.5 Trivial Languages 80

5.3.6 Locally Threshold Testable 81

5.3.7 $\textsc{fo}^2[<]$ and $\mathscr{G}$-Triviality 82

5.3.8 Generalized Locally Testable 84

5.3.9 $\textsc{fo}^2[<,\lhd]$ and Local $\mathscr{G}$-Triviality 85

5.3.10 $\textsc{fo}^2[<,\mathrm{bet}]$: Adding Betweenness 85

5.3.11 Languages Locally $\mathscr{J}$-Trivial 86

5.3.12 Generalized Locally $\mathscr{J}$-Trivial Languages 89

5.4 Conclusions 91

6 **Tree Recognition with Strongly Directed Hypergraphs** **93**

6.1 Background 94

6.1.1 Trees 94

6.1.2 Finite-State Acceptors 95

6.1.3 Directed Graphs and Extensions Thereof 96

6.2 Strongly Directed Hypergraphs 98

6.3 Decisions and Operations 101

6.3.1 Reachability and Satisfiability 102

6.3.2 Determinization 102

6.3.3 Minimization 103

6.3.4 Completion and Trimming 105

6.3.5 Finiteness 106

6.3.6 Boolean Operations 107

6.4 Conclusions 109

7 Accumulators and the Problems They Bring 111

7.1 Parsing as a Monoid 111

7.2 Going Further: Turing Completeness 116

7.3 Conclusions 119

8 Conclusions 121

Bibliography 125

List of Figures

2.1 Precedence and successor models of “ababc”. 13

2.2 Two three-windows of “ababc” for the same factor. 16

2.3 Even-a as a string acceptor. 19

3.1 Word models for “ababc”. 24

3.2 3-factors of “$\rtimes$ababc$\ltimes$”. 25

3.3 A canonical DFA for which d is a nonsalient symbol. 35

3.4 The powerset graph of Figure 3.3. 36

3.5 Canonical AAA for the factor “xyx” under $<$. 38

3.6 Canonical automata for head-anchored factors. 39

3.7 Constructing “xyx” as an AAA under $\lhd$. 40

3.8 Relativizing “xyx” to $\mathrm{T} = \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$. 40

3.9 The syntactic semigroup of Figure 3.3. 47

4.1 The tier-successor relation on “lokalis”. 60

4.2 Space requirements for learning over a binary alphabet. 67

5.1 Every element a of a finite semigroup S eventually generates a group. 71

5.2 Hierarchy of inclusion for algebraically characterized classes. 90

6.1 Accepting a tree. 96

6.2 Even-a as a string acceptor. 97

6.3 S $\rightarrow$ aSb and S $\rightarrow$ ab as an unlabeled directed hypergraph. 98

6.4 S $\rightarrow$ aSb and S $\rightarrow$ ab as a labeled strongly directed hypergraph. 99

6.5 A tabular representation of Figure 6.4. 100

6.6 A tree acceptor for Boolean expressions over two variables. 100

6.7 The tabular representation of Figure 6.6. 101

6.8 A strongly directed hypergraph representing Figure 6.2. 101

6.9 A nonminimal DBFTA. 104

6.10 A minimal form of Figure 6.9. 105

6.11 Instantiating ?⃝ for a rank-3 S. 106

6.12 A DBFTA and its associated connection graph. 107

7.1 Parses for “()()” and “(())”. 112

7.2 Basic shape of the parsing matrix. 113

7.3 Parsed “$_0$the$_1$girl$_2$saw$_3$the$_4$crow$_5$”. 114

7.4 Parsed “$_0$the$_1$girl$_2$saw$_3$the$_4$crow$_5$with$_6$the$_7$binoculars$_8$”. 115

List of Tables

4.1 Augmented subsequences of “cabacba”. 62

4.2 Words sufficient to learn Latin liquid dissimilation. 65

5.1 A summary of classes and their characterizations. 90

Chapter 1: Introduction

Numerous types of constraints are used in description of linguistic patterns. I explore the types of patterns definable with a constant memory bound over both strings and trees. With a primary focus on phonology and strings, I discuss how learnability considerations may provide reason to prefer some constraints over others. Methods for extracting constraints are provided, both from data (learning) and from automata (factoring). These may help in finding descriptions extensionally equivalent to a given pattern. Also, if a typologically predictive theory is desired then care must be taken to avoid formalisms that can define all computable functions.

Finite-state machines formalize the notion of a constant memory bound and have been widely used for decades in both computer science (Thompson, 1968) and linguistics. Using different properties of the machines or of the formal languages they represent, we can discover a great deal about languages, including their complexity (Rogers et al., 2012) or how they relate to one another (Clark and Roberts, 1993). Constraint-based analysis have proven particularly useful (Lambert and Rogers, 2019), so I aim to describe both factorization methods, for cases in which one is presented with a structure, and learning methods, for cases in which one is presented with data.

In classification, algebraic characterizations are often used because they can just as easily affirm or deny the inclusion of a language in a class. But also language-theoretic characterizations are useful, because they are machine agnostic. I aim to provide both types of characterizations for some classes which previously have received no such treatment, specifically the tier-based classes and the (strongly) costrict classes. Further, tree languages and finite-state methods thereon have traditionally been treated as entirely distinct from tree languages, so I aim to unify the analyses.

Acceptors are not everything, however. Often in linguistics we wish to discuss transformations from one structure to another, or more generally to explain how related structures relate. There are several approaches in wide use, including logical transductions (Courcelle, 1994), semiring-based analyses (Lothaire, 2005), and finite-state systems with outputs on the edges (Mohri, 1997). The last of these is really a special case of the second. I aim to show that the semiring-based analysis is too powerful.

In general, I aim to unify finite-state methods for languages and relations over strings and trees, all while avoiding the issue of unbounded computational power. How do we determine which constraints they satisfy? In this work, I begin this unification by discussing the methods used on string languages and by proposing a unifying graphlike structure for string and tree languages. Rather than discuss the structure of string languages or of tree languages or of relations, I would like to discuss simply the structure of structure.

In short, I am tackling a lack of unity and completeness in the formal description of linguistic patterns. The result of this work will provide a firm mathematical and computational basis for linguistic description and for models of language learning, in addition to improving pedagogy and tools relating to constraint-based descriptions of languages.

1.1 Review of the Literature

Kleene (1956) introduces the regular languages and associates them with finite-state automata. McNaughton and Papert (1971) discuss formal languages built around local dependencies: the strictly local and locally testable classes. These languages have some nice properties: they are efficiently learnable, efficiently testable, and closed under intersection (and thus usable as constraints) (Heinz, 2010b; Rogers et al., 2012). Indeed, these classes were studied for their relative simplicity compared to the regular languages, requiring no modulo-counting mechanism. The locally threshold testable class of Beauquier and Pin (1989) lies between regular and locally testable allowing for counting of local structures. Local constraints can't handle everything, however, and natural language is full of patterns outside of these classes, such as the stress pattern of Yidin (Goedemans et al., 2015; Lambert and Rogers, 2019), the asymmetric sibilant harmony of Sarcee (Heinz, 2010a), or the tone plateauing of Luganda (Hyman and Katamba, 1993). None of these are representable even by locally threshold testable descriptions. They can be analyzed by star-free languages (also known as group-free, locally testable with order, or noncounting), but often a simpler analysis is possible.

All of these difficult patterns invoke long-distance dependencies. The piecewise testable languages of Simon (1975) directly encode this type of dependency. Mirroring the strictly local restriction of the locally testable languages, the piecewise testable class can be restricted to a strictly piecewise class (Rogers et al., 2010, see also Haines, 1969). There is no distinct piecewise threshold testable class; it is equivalent to piecewise testable (Lambert et al., 2021).

Another way of representing some types of long-distance dependencies, based on popular feature geometry models such as those discussed by McCarthy (1988), is the class of tier-based strictly local languages described by Heinz et al. (2011). There is no corresponding tier-based strictly piecewise class, because the general precedence relation is not strengthened or weakened by the ability to ignore specific symbols. The naturalness and power of tier-based descriptions has caused them to be widely used and extended (McMullin, 2016; Jardine and McMullin, 2017; Aksënova and Deshmukh, 2018; De Santo and Graf, 2019; Lambert, 2021a). Mayer and Major (2018) even discuss a hypothesis that all phonological patterns are tier-based strictly local, then go on to provide a counterexample. This should come as no surprise; the tone plateauing of Luganda is also outside of this class, unless we assume a representation that removes the unrepresentable long-distance dependencies.

My opinion is that it is a mistake to believe that any given subregular class could act as a universal descriptor. Instead, a focus on classification is important, as each class has a set of associated properties, and that can tell us a lot about the patterns we see. Patterns may be described in several equivalent ways, perhaps belonging to several classes and sharing all of their properties. If given data, or a prosodic description of a pattern, it may be easy to discover counterexamples to some class-specific language-based characterization. For instance, the strictly piecewise languages are closed under deletion (Rogers et al., 2010), so if a language in this class contained a word CVCV, it would necessarily also contain a word CVC. These language-theoretic characterizations provide an easy way to say that a language cannot be in the class, but a guarantee of inclusion may be more difficult to provide. A grammatical-formalism based analysis has the opposite problem: providing a grammar is an easy way to guarantee inclusion, but it may be difficult to prove that no such grammar is possible.

McNaughton and Papert (1971) discuss the use of the syntactic monoid of a language to determine if that language is star-free. This algebraic structure is derived from the canonical acceptor for the language in question. Algebraic procedures for deciding membership in the locally testable class (Brzozowski and Simon, 1973; McNaughton, 1974), the locally threshold testable class (Beauquier and Pin, 1989), and the piecewise testable class (Simon, 1975) also exist. For those classes testable in the strict sense, an algebraic approach finds an issue: an automaton and its complement share a syntactic monoid, but these classes are not closed under complementation. More information is necessary beyond the structure itself, as shown by De Luca and Restivo (1980) for the strictly local class, or by Fu et al. (2011) for the strictly piecewise class. Lambert (2021b) describes a general algebraic charcterization for tier-based languages. Indeed, several subregular classes have algebraic decision procedures, including the class defined by the fragment of first-order logic restricted to two variables and general precedence (Thérien and Wilke, 1998) and possibly betweenness (Krebs et al., 2020).

A pattern may be defined by zero or more constraints over the set of all possible words. Rather than classifying a pattern in its entirety, it may be useful to factor it into simple constraints and classify those (Lambert and Rogers, 2019). Leaving the pattern in its factored state can result in a smaller state space and easier processing (Heinz and Rogers, 2013). Rogers and Lambert (2019b) describe a mechanism for extracting some types of subregular constraints from finite-state automata. Lambert (2021b) shows how factoring can be extended to tier-based classes, but a persistent problem plagues tier-based analyses: other constraints easily interfere with the tier selection process (Lambert, 2021a).

As with classification, it may be desired to determine which constraints are satisfied by some data without a neat, complete description of the pattern. This is the learning problem. Gold (1967) proposed several learning frameworks, and one in particular has caught on: learnability in the limit from positive data. The restriction to only positive data both provides a more tightly restricted learning paradigm and, as Yang (2015) discusses, may more accurately reflect the acquisition process of natural language. Most if not all of the subregular classes are learnable in the limit from positive data using some variant of Heinz's (2010b) string extension learning (Lambert et al., 2021; Lambert, 2021a). One may go so far as to suggest that without an effective learning procedure, a class is useless in description of natural language phenomena. The robustness of the learning in the presence of sparse data may also be of concern.

The discussion so far has dealt only with string languages definable by a state machine of finitely many states. Syntax cannot be described this way. Syntax, like many other hierarchical systems, deals in trees. Rogers (1997) showed that context free grammars, while producing superregular string yields, are only strictly 2-local over trees. Graf (2018) shows that, in at least some sense, the operations fundamental to Minimalist syntax are tier-based strictly local on trees. The question becomes "how do we classify tree languages?" Klein and Manning (2004) and Huang and Chiang (2005) consider state machines based on a chart parse such as a CKY table, where the number of states grows with the size of the input. True finite-state machines also appear; Gécseg and Steinby (1984) and Comon et al. (2007) both detail finite-state analyses wherein a recursive procedure assigns a state to each subtree before deciding the state for the tree as a whole. It seems that none have constructed graph-based analyses of problems with respect to these tree automata, so this problem is explored here. On the topic of trees, algebraic characterizations are indispensible in classifying string languages. But the syntactic monoids of context free languages are infinite, and may not have such nice properties. Clark (2015) discusses a different algebraic system over strings which classifies the context-free languages, and a few algebraic systems have been proposed for dealing with tree languages (Germain and Pallo, 2000; Steinby, 2015), but it remains unclear how these connect to a subregular hierarchy for trees.

Finite state acceptors can also be extended by associating outputs with their edges in order to describe transformations, sometimes referred to as transductions. If each input symbol is required to be associated with some sequence of output symbols, which are concatenated as computation progresses, then the rational relations are described (Frougny and Sakarovitch, 1993, see also Rabin and Scott, 1959). Deterministic relations (functions) in this class are called "subsequential", and are frequently used in phonology (Chandlee, 2014). Indeed (Chandlee, 2014) describes subclasses of the subsequential functions, input strictly local and output strictly local, to account for certain phonological properties. Tonal phonology and templatic morphology are described by input strictly local functions over multiple tapes (Dolatian and Rawski, 2020; Rawski and Dolatian, 2020), and tier-based functions have been proposed (Burness and McMullin, 2019). Another common approach, which uses semirings to unify acceptors, transducers, weighted automata, and more, is described by Lothaire (2005). This semiring approach essentially augments the state machine with an accumulator in some monoid. I show that the power of such an accumulator is unrestricted. The approach with less freedom in structure yields a more restrictive set of relations. Courcelle (1994) describes logical transductions, which abstract away from the machine. Classes of string languages may be related both to a particular kind of algebraic property and to some particular kind of logical formalism (Rogers and Lambert, 2019a), so the same ought to hold for classes of functions. Unbounded deletion prevents input strictly local functions from aligning with a kind of quantifier-free logic, but restrictions on functions based on origin information (Bojańczyk, 2014; Bojańczyk et al., 2017) may be useful in closing the gaps.

1.2 Outline of the Dissertation

Chapter 2 introduces the core concepts that will be used throughout this work, including (sub)regular languages, finite-state automata, model theory, and factors. Beyond that, this dissertation is structured such that the remaining chapters can be read as independent works in their own right.

The piecewise-local subregular hierarchy is a partial order of language classes which can be described via the successor or general precedence relations using at most monadic second-order logic. Tier-based languages are defined by the transitive reduct of a particular restriction of the general precedence relation. A subset of the piecewise-local classes form a grid, where one axis describes the types of factors in use and the other describes the kinds of distinctions made. Other classes fit less neatly into such a paradigm. Chapter 3 discusses classification of regular languages into the classes of the piecewise-local subregular hierarchy, including algebraic, automata-theoretic, language-theoretic, and model-theoretic approaches to the problem. In particular, the complements of strictly langugaes are newly characterized, and tier-based extensions are provided for all classes of the hierarchy, extending the tier-based strictly local class. Chapter 5 incorporates classes from the computer science literature into the hierarchy, namely those classes based on definability with first-order logic restricted to two variables and several other classes characterized algebraically.

While chapter 3 includes brief discussion on extracting constraints from automata, Chapter 4 incorporates discussion on constraint extraction from data (learning). This problem is explored through a generalization of string extension learners, extending the original definition to account for different possible interpretations of grammars, and a learning algorithm for tier-based languages is defined.

There are many ways to represent string acceptors. One might use the potentially infinite set describing the target language, some kind of grammar, or a state machine. These state machines are represented by graphs, but properly superregular languages require infinitely many states. Context free languages can, however, be recognized with finitely many states using a tree acceptor rather than a string acceptor. Chapter 6 introduces strongly directed hypergraphs in order to represent tree acceptors as direct generalizations of string acceptors. The Boolean operations are described in much the same way as the graph-based algorithms for strings. Some decision problems require less structure, and these reductions are described.

Throughout these chapters it is discussed that algebraic structure is immensely useful in complexity classification. Some have expressed interest in describing certain phenomena using semiring-based transducers, which essentially are finite-state acceptors augmented with a monoidal accumulator. Chapter 7 discusses the expressive power of such a structure. First I show that a CKY-style chart parse can be expressed with a monoid. Then I generalize, demonstrating that a run of a Turing machine can also be expressed as a monoid. In order to maintain a truly finite amount of state, one may demand the use of a finite monoid, but this prohibits representing even the identity function. A fundamental question remains: what kinds of restrictions should a monoidal accumulator satisfy? My answer is that the monoidal accumulator is the wrong approach.

The final chapter concludes the work by reiterating the goals of a structure-based approach to linguistic analysis and stating directions for future work.

Chapter 2: Formal Languages and String Acceptors

This chapter introduces formal languages and other concepts that will be used throughout the text, including finite-state automata and finite model theory. If a language can be associated with some fixed memory bound under which any possible word can be tested for membership, then that language is regular. The piecewise-local subregular hierarchy, which will be explored further in Chapter 3, is a collection of classes of regular languages, where each class imposes its own additional constraints.

2.1 Formal Language Theory

A word is a sequence of symbols drawn from some alphabet, typically written Σ. The set of all possible words is Σ^*, and the set of nonempty words is Σ^+. A formal language is nothing more than a possibly infinite set of words. Such a set may also be referred to as a pattern or, when considering intersections of multiple patterns, a constraint. Different classes of formal languages are defined based on what mechanisms are needed in order to decide whether a word belongs to the language.

The locally testable and strictly local languages discussed by McNaughton and Papert (1971) can be recognized by a fixed-width left-to-right scanner (see also Beauquier and Pin, 1991). Several language classes admit more than one characterization. For example, the locally threshold testable class of Beauquier and Pin (1989) can be described as all and only those language recognizable by such a scanner where each possible factor has its attestation count incremented (up to some threshold) each time it is encountered and the acceptability of the word is determined by the collection of counts. Or it could be described as all and only those languages first-order definable with successor Thomas (1982). Each of these characterizations tells us something about the languages in the class.

2.2 Finite Model Theory

Concepts from finite model theory provide a uniform way to describe relational structures and their parts in logical terms (see Libkin, 2004 for a thorough introduction). Applying these concepts to linguistic structure is not a new idea, with applications to syntax by Rogers (1996, 1998) beginning to popularize the approach.

A relational word model consists of a **domain**, $\mathcal{D}$, which is isomorphic to an initial segment $\{1, \ldots, n\}$ of the nonzero natural numbers and represents positions in the word, as well as a collection of relations, $R_i \subseteq \mathcal{D}^{a_i}$, each of which has its own arity a_i.

$$\mathscr{M}(w) = \langle \mathcal{D}; R_i \rangle.$$

Generally we assume that a model consists of at least one **ordering** relation, as well as one or more unary **labeling** relations that partition the domain. Additional relations of any arity are of course permitted. The assumption of a partition is nonrestrictive; one can convert a model whose labeling relations do not form a partition of the domain into a partitioned normal form by using the powerset of these relations instead. One simple example of an ordering relation is that of general precedence ($<$), where $a < b$ if and only if (iff) the domain element a occurs anywhere before b.

The immediate successor relation that defines the local branch of the subregular hierarchy can be derived in first-order logic from general precedence.

$$x \triangleleft y \ \triangleq \ (x < y) \wedge \neg(\exists z)[x < z < y].$$

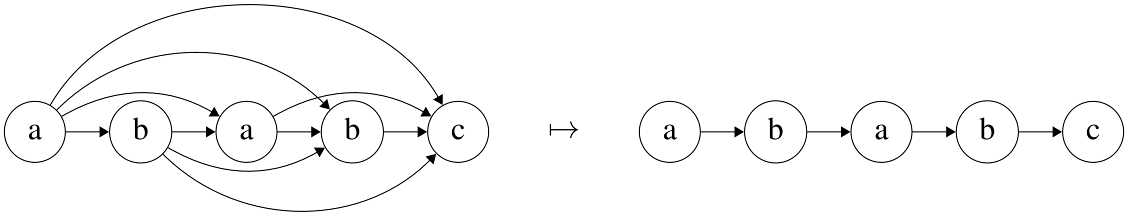


Figure 2.1: Precedence and successor models of “ababc”.

This is simply the transitive reduction of this general precedence relation. An example word model is shown in Figure 2.1.

Now that we have a notion of a structure, we turn to discussion of contained structures.[1] Following Lambert and Rogers (2020), two concepts are distinguished: a **factor**, which is a connected structure contained within a model, and a **window**, which is a structured collection of domain elements from which a factor may be derived. We begin by defining a window.

Given a (homogeneous) relation R of arity $a \geqslant 2$, i.e. $R\colon \mathcal{D}^a$, its a-windows are defined by the set

$$\mathscr{W}_a^R \triangleq \Big\{ \big\{ \langle \overset{i}{x}_i, \overset{i+1}{x}_{i+1} \rangle \colon 1 \leqslant i < a \big\} \colon \langle x_1, \ldots, x_a \rangle \in R \Big\}.$$

This effectively turns each tuple in the relation into a sequence of overlapping pairs that represent the edges of a (linear) directed graph version of that tuple. Each node in this graph is labeled by not only the domain element itself, but also an index so that cycles in the structure do not translate into cycles in the window. For instance, if 1 is a domain element and $\langle 1, 1 \rangle$ appears in R, the only

[1] Because our notion of structural containment is distinct from the standard model-theoretic notion of substructures, we are careful to avoid that term.

2-window in the set of 2-windows that corresponds to this relation is

$$\big\{\langle \overset{1}{1}, \overset{2}{1}\rangle\big\}.$$

Discussing smaller windows is simple. Any connected subgraph of an a-window is also a window, of size equal to the number of nodes it contains.

For windows of size larger than the arity of the relation from which they are defined, we can use an inductive definition:

$$\begin{aligned}
\mathscr{W}^R_{k+1} \triangleq \Big\{ A \cup \langle \overset{j_{a-1}}{x_{a-1}}, \overset{k+1}{x_a}\rangle \colon A \in \mathscr{W}^R_k \text{ and } & \langle x_1, \ldots, x_a\rangle \in R \\
\text{and } & \{j_1, \ldots, j_{a-1}\} \subseteq \{1, \ldots, k\} \\
\text{and } & \{\langle \overset{j_i}{x_i}, \overset{j_{i+1}}{x_{i+1}}\rangle \colon 1 \leqslant i < a-1\} \subseteq A \\
\text{and } & (\exists y, \ell)[\langle \overset{j_{a-1}}{x_{a-1}}, \overset{\ell}{y}\rangle \in A \text{ or } \langle \overset{\ell}{y}, \overset{j_{a-1}}{x_{a-1}}\rangle \in A] \\
\text{and } & \big(\forall j_a \in \{1, \ldots, k\}\big)\big[\langle \overset{j_{a-1}}{x_{a-1}}, \overset{j_a}{x_a}\rangle \notin A\big]\Big\}.
\end{aligned}$$

The conditions on the first line select a k-window and an element of the relation. The second line selects $a-1$ indices. The third line, which is never relevant for a binary relation, ensures that these indices form a path from the first to the last, and that this path is labeled by the appropriate domain elements. The fourth line accounts for binary relations, simply asserting that the selected index corresponds to an appropriate domain element. And finally the fifth ensures that edges in the model may only be repeated in the case of cycles.

In short, for each k-window, we find a linear subgraph (a path) that maps to the first $a-1$ elements of a tuple in R, then add an edge from the final node of this path to a newly constructed node representing the final domain element from that tuple.

The conditions assert that adding this new node does not simply repeat the construction of an already-existing path, while still allowing cycles to be iterated without bound. For example, given the 2-window described previously, the only valid 3-window formed from the same $\langle 1, 1\rangle$ tuple is

$$\{\langle \overset{1}{1}, \overset{2}{1}\rangle, \langle \overset{2}{1}, \overset{3}{1}\rangle\}.$$

Both index 1 and index 2 provide valid attachment points for a $\langle 1, 1\rangle$ edge, but this edge has already been followed from index 1, so that attachment is ruled out by the condition on the fifth line. The result of this induction is that a window is a rooted, connected, acyclic graph of indexed domain elements, where the root is the unique node of in-degree zero.

Although only binary relations will be discussed in this work, this definition applies more generally. Consider $R_3 = \langle 1, 2, 3\rangle, \langle 1, 2, 4\rangle, \langle 2, 4, 5\rangle, \langle 3, 4, 5\rangle$. Two pairs can be chained: $\langle 2, 4, 5\rangle$ can overlay the right of $\langle 1, 2, 4\rangle$, or $\langle 1, 2, 4\rangle$ and $\langle 1, 2, 3\rangle$ can be overlaid at their left portions. So the only 4-windows of R_3 are

$$\left\{\langle \overset{1}{1}, \overset{2}{2}\rangle, \langle \overset{2}{2}, \overset{3}{4}\rangle, \langle \overset{3}{4}, \overset{4}{5}\rangle\right\} \text{ and } \left\{\langle \overset{1}{1}, \overset{2}{2}\rangle, \langle \overset{2}{2}, \overset{3}{4}\rangle, \langle \overset{2}{2}, \overset{4}{3}\rangle\right\} \text{ and } \left\{\langle \overset{1}{1}, \overset{2}{2}\rangle, \langle \overset{2}{2}, \overset{3}{3}\rangle, \langle \overset{2}{2}, \overset{4}{4}\rangle\right\}.$$

It is possible that an alternative definition requiring less overlap might be preferred. Notice though that the latter two windows are identical graphs when the indices are ignored.

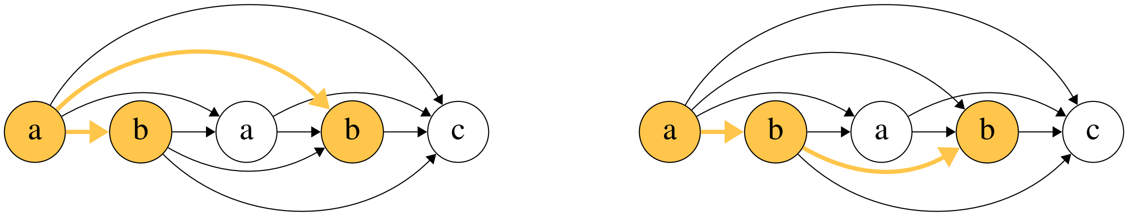


Figure 2.2: Two three-windows of “ababc” for the same factor.

In general, there can be several windows that correspond to the same contained structure. Looking only at the domain elements represented unifies these multiple representations: the **factor at** a window x in the word model m (written $[\![x]\!]_m$) is the restriction of m to the domain elements in x. For instance, consider the signature

$$\mathscr{M}^{\triangleleft} = \langle \mathcal{D}; <, \mathrm{a}, \mathrm{b}, \mathrm{c} \rangle$$

and a word model over this signature

$$m = \Big\langle \{1,2,3,4,5\}; \{\langle x,y\rangle : x, y \in \mathcal{D} \text{ and } x < y\}, \{1,3\}, \{2,4\}, \{5\} \Big\rangle,$$

If we have a window x such that the domain elements included in x are all and only 1, 2, and 4, such as either of those in Figure 2.2, then the factor at x is the corresponding restriction

$$[\![x]\!]_m = m \restriction x = \Big\langle \{1,2,4\}; \{\langle 1,2\rangle, \langle 1,4\rangle, \langle 2,4\rangle\}, \{1\}, \{2,4\}, \varnothing \Big\rangle.$$

The set of all k-factors of a model m is the set

$$\mathscr{F}_k(m) \triangleq \left\{[\![x]\!]_m \colon x \in \mathscr{W}_k\right\}.$$

where the windows are built over the ordering relations of m.

The word models shown to this point have been without explicit indication of domain boundaries. Often, however, we wish to consider models in which these boundaries are explicit, which we call **anchored models**. These can be formed by augmenting a model with new positions, self-related under all ordering relations, that are labeled “$\rtimes$” and “$\ltimes$” for head and tail boundaries, respectively. This self-relation allows words shorter than k to be captured by k-windows without special treatment.

One might notice that a model that has a smaller number of domain elements than the arity of its ordering relation might have no factors at all by these definitions. While this is never a problem for anchored models, one might consider alternative constructions when using nonanchored models. The simplest is to construct the factors of the anchored models and then strip away the domain boundaries from the result. In any case, the use of anchored word models will be assumed throughout this text.

2.3 Graphs and Finite-State Automata

A finite-state acceptor is a mechanism for deciding whether a structure belongs to a given set by reading the structure in some reasonable order and traversing through a finite set Q of states. Such an acceptor for strings over a finite alphabet Σ consists of its set of states, a transition function

$\delta\colon \Sigma \times Q \to Q$, an initial state q_0, and a set $F \subseteq Q$ of accepting states. This is typically written as the five-tuple $\langle \Sigma, Q, \delta, q_0, F \rangle$. For instance, the following represents the set of strings over the set $\{a, b\}$ that contain an even number of occurrences of a:

$$\mathcal{A} = \langle \{a, b\}, \{q_1, q_2\}, \delta, q_1, \{q_1\} \rangle$$

$$\delta(a, q_1) = q_2$$

$$\delta(a, q_2) = q_1$$

$$\delta(b, q_1) = q_1$$

$$\delta(b, q_2) = q_2$$

There are constructive proofs that these string acceptors are expressively equivalent to regular expressions and thus recognize all and only regular string languages (McNaughton and Yamada, 1960). Strings are canonically read left-to-right, but this class is closed under reversal so it follows that a right-to-left automaton would be equally expressive. If δ is a function, such an acceptor is called a deterministic finite-state automaton (DFA).

An automaton representing all and only those strings which contain an even number of occurrences of a is shown as a labeled directed graph in Figure 2.3. The alphabet is $\{a, b\}$ Each state of the acceptor is a node, and each element $\langle \sigma, q, r \rangle$ of δ is represented by an edge from q to r labeled σ. The accepting states are marked as such, and the initial state is denoted by an arrow from nowhere. A word w is accepted iff there is some path $q_o \to \cdots \to q_f$ for some $q_f \in F$ whose labels spell out w. For instance, “abba” is accepted by the DFA of Figure 2.3.

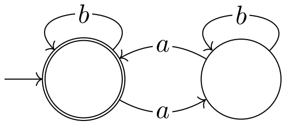


Figure 2.3: A finite-state string acceptor: doubly-outlined states are accepting, and the initial state is marked by the arrow from nowhere. All and only those strings which contain an even number of occurrences of a are accepted.

2.4 Notation

A function from a domain $\mathcal{D}$ to a codomain $\mathcal{D}'$ is written $f\colon \mathcal{D} \to \mathcal{D}'$. The set of natural numbers is written $\mathbb{N}$. As a special case reflecting traditional notation for sequences, the value of a function $f\colon \mathbb{N} \to \mathcal{D}'$ at n is written f_n.

Chapter 3: Classifying and Factoring Tier-Based Extensions of the Subregular Hierarchy

The piecewise-local subregular hierarchy, henceforth referred to as simply **the subregular hierarchy**, has been extensively studied for decades, with the local branch introduced by McNaughton and Papert (1971) and the piecewise branch stemming from Simon (1975). Local constraints are, as the name implies, good at capturing dependencies based on adjacent events, and can do so with even the simplest logics. Even some long-distance dependencies can be captured, such as “A and B do not occur in the same word”, but there is no notion of directionality here. Piecewise constraints fall on the other extreme, easily representing certain types of long-distance dependencies but requiring at least first-order logic to be able to refer to adjacent events at all.

In order to more simply state some types of long-distance dependencies, and to account for some that piecewise constraints cannot, a third branch came into existence with the tier-based strictly local (TSL) class imposing adjacency on distant parts of a string (Heinz et al., 2011). A TSL description works by relativizing the concept of adjacency over some subset of the alphabet, referred to as the tier alphabet. Symbols outside this subset are ignored entirely.

Linguistic interest in the TSL class stems from its usefulness in describing long-distance dependencies, especially those that strictly piecewise constraints cannot handle such as blocked harmony patterns (Heinz et al., 2011; McMullin, 2016). For example, the liquid dissimilation pattern of Latin (Cser, 2010) is a sort of blocked harmony pattern, shown to be 2-TSL by McMullin (2016). This pattern can be shown to be strictly star-free when restricted to the local or piecewise branches of the subregular hierarchy. However, another important consideration for linguistics is learnability, and the star-free class is not learnable (Gold, 1967). On the other hand, 2-TSL has been shown to be effectively learnable both by humans (McMullin, 2016) and by machines (Jardine and Heinz, 2016), and k-TSL for arbitrary k has been shown to be learnable as well (Jardine and McMullin, 2017).

The main formal result of Heinz et al. (2011) was a language-theoretic proof that TSL is a subclass of star-free. Originally TSL was defined by applying an erasing homomorphism to a string(set), projecting it to strings formed from the tier alphabet, then applying a strictly local filter to the result. This operational perspective is useful in describing the solution, but it can mask some insights. To provide more clarity, Lambert and Rogers (2020) provide an equivalent alternative definition based on model theory and a new class of ordering relations, and from this they develop language- and automata-theoretic **characterizations** of the class. A characterization of a class is a property such that all and only those sets that have this property are in the class.

The present work extends this further, characterizing not only the TSL class but also relativized variants (introduced here) of the other classes in the subregular hierarchy. These characterizations also go further, including algebraic as well as automata-, language-, and model-theoretic results.

We begin in section 3.1 with an overview of the model-theoretic concepts that will be used, then section 3.2 provides definitions as well as model- and language-theoretic characterizations for all of the relativized classes. Section 3.3 provides automata-theoretic characterizations for some of the classes, deferring the rest to section 3.5 in which automata are converted to an algebraic structure. In this latter section, algebraic characterizations are given for each new class, primarily synthesizing classical results and applying them to the new structures. Algebraic characterizations are explored even further in chapter 5. Closure properties are proved in section 3.4, using properties of automata to prove that some closures do hold, while using the language-theoretic characterizations provided earlier to demonstrate that some other potential properties do not.

3.1 Model Theoretic Descriptions

In this section we discuss a model-theoretic treatment of relativized adjacency. The relation defined here is then used to define variants of each class of the subregular hierarchy, where the relativized variant of the strictly local class is identical to the tier-based strictly local class of Heinz et al. (2011).

Recall that the successor relation is the transitive reduction of the general precedence relation. Instead of reduction, we might consider restricting the domain to all and only those elements that satisfy a certain predicate φ.

$$x <^{\varphi} y \triangleq \varphi(x) \wedge \varphi(y) \wedge (x < y).$$

This is essentially the general precedence relation on the model's projection to those elements that satisfy φ. We can of course combine these to obtain the reduction of this restriction,

$$x \lhd^{\varphi} y \triangleq (x <^{\varphi} y) \wedge \neg(\exists z)[x <^{\varphi} z <^{\varphi} y],$$

which defines a relativized successor relation. Given some alphabet Σ and some set of salient symbols $\mathrm{T} \subseteq \Sigma$, the predicate $\varphi(x) = \mathrm{T}(x) = \bigvee_{\tau \in \mathrm{T}} \tau(x)$ results in a relation that acts as if it were the successor relation on the model's projection to T, reminiscent of the original definition of the $\mathrm{TSL}^{\mathrm{T}}$ class. We refer to this specific kind of relativization as **projective relativization**. The

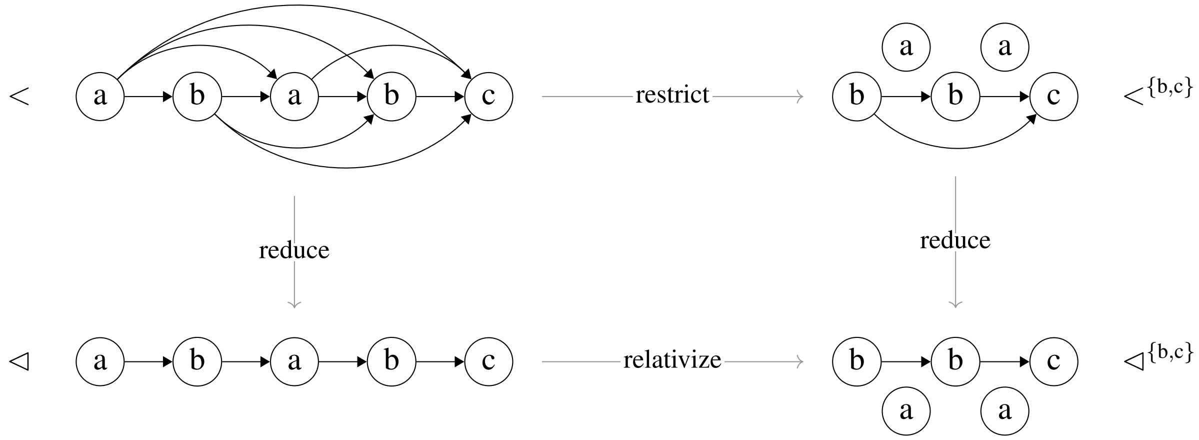


Figure 3.1: Word models for "ababc" using general precedence, immediate successor, and relativized variants of each, showing the relationships among these relations. Domain elements that are not ordered are pulled aside from the structure. In these examples, the alphabet is $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, and the salient symbols for the relativized relations are $\mathrm{T} = \{\mathrm{b}, \mathrm{c}\}$

particular definitions used here guarantee that the factors of a model under a projectively relativized relation are exactly those under the corresponding nonrelativized one of the model's projection.

Figure 3.1 shows how the general precedence and immediate successor relations, as well as their relativized variants, relate to one another. These relationships do not only hold for these relations. In fact any binary relation whose transitive closure is antisymmetric may be relativized by taking the transitive reduction of a restriction of its transitive closure. The antisymmetry requirement ensures uniqueness of the result (Aho et al., 1972). Throughout this text we will consider only projective relativization, though with appropriate choice of φ the more general treatment can be shown to capture the structure-sensitive tier-based strictly local class of De Santo and Graf (2019) or the domain- and interval-based strictly piecewise classes of Graf (2017). One important property specific to projective relativization is that the unordered elements truly have no effect on the ordered ones. Whether a domain element is included in the restriction is decided entirely by the unary relation that labels that point, and so the unordered elements can be freely removed, shuffled, or

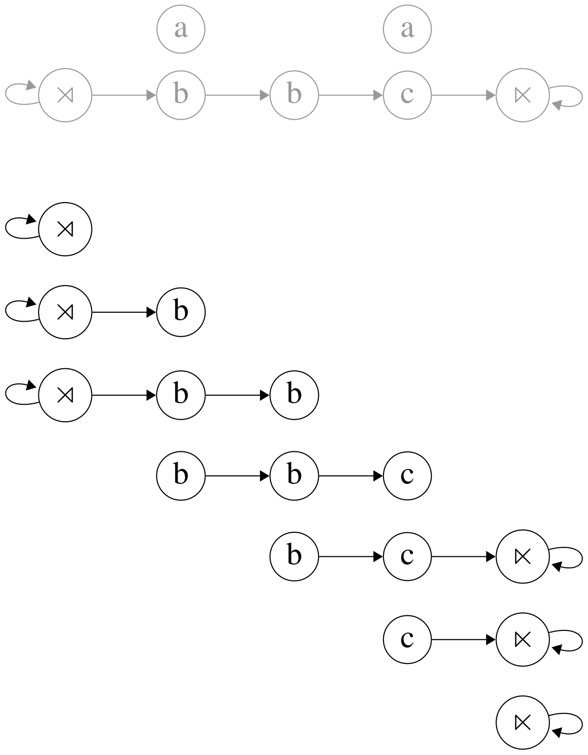


Figure 3.2: All 3-factors of "$\rtimes$ababc$\ltimes$" under the $\lhd^{\{\text{b,c}\}}$ relation. Factors that appear shorter than this are formed from windows that repeat the boundary symbols.

inserted at any point.

For any projectively relativized relation, we assume for notational convenience that the boundary symbols $\rtimes$ and $\ltimes$ are considered salient if they are present in the model. So rather than writing $\lhd^{\{\rtimes,\text{b,c},\ltimes\}}$ we simply write $\lhd^{\{\text{b,c}\}}$ instead. Figure 3.2 shows an anchored word model for "ababc" under the $\lhd^{\{\text{b,c}\}}$ relation along with all of its nonempty 3-factors. Because the domain boundaries are self-related, assuming the domain elements of "$\rtimes$ababc$\ltimes$" are 1 through 7 in order, the windows

$$\{\langle \overset{1}{1}, \overset{2}{1}\rangle, \langle \overset{2}{1}, \overset{3}{3}\rangle\} \quad \text{and} \quad \{\langle \overset{1}{1}, \overset{2}{1}\rangle, \langle \overset{1}{1}, \overset{3}{3}\rangle\}$$

both refer to the relativized prefix "$\rtimes$b" of this string. (Either instance of domain element 1 is a valid attachment point for a $\langle 1, 3\rangle$ edge.) The “a” elements are unordered and do not occur in any factor.

3.2 Language-Theoretic Characterizations

A grammar is some representation of a mechanism by which the membership of a string in a stringset may be decided. A class of grammars is denoted by $\mathbb{G}$. The characteristic function $\mathbb{1}\colon \mathbb{G}\times\mathscr{M}\to\mathbb{B}$ is

$$\mathbb{1}_G(m) \triangleq \begin{cases} \top & \text{if } m \text{ satisfies } G,\\ \bot & \text{otherwise.}\end{cases}$$

Here, $\mathbb{B}$ represents the binary Boolean ring, $\top$ is true, $\bot$ is false. Functions of more than one argument are sometimes written with their first argument as a subscript; $\mathbb{1}_G$ can be thought of as the partial application of the curried form of $\mathbb{1}$. The stringset represented by G is the set of all and only those strings whose models satisfy it:

$$\mathscr{L}(G) \triangleq \bigl\{w\colon \mathbb{1}_G\bigl(\mathscr{M}(w)\bigr)\bigr\}.$$

Two grammars G_1 and G_2 are equivalent iff they are extensionally equal, that is, $\mathscr{L}(G_1) = \mathscr{L}(G_2)$.

3.2.1 Strict locality

The original definition of the TSL class from Heinz et al. (2011) was operational. A stringset L is k-TSL$^{\mathrm{T}}$ iff there exists a k-SL grammar such that L contains all and only those strings whose projection to T satisfy this grammar.

A grammar for a k-SL stringset is simply a subset of $\mathscr{F}_k(\Sigma^*)$, that is, a set of k-factors, where an anchored model m satisfies G iff each of its k-factors occurs in G (McNaughton and Papert, 1971):

$$\mathbb{1}_G(m) \;=\; \mathscr{F}_k^{\triangleleft}(m) \subseteq G.$$

Because the symbols not in T never appear in any factors under $<^{\mathrm{T}}$ or $\triangleleft^{\mathrm{T}}$, any two strings with the same projection to T will have the same set of factors under these relations. Specifically, if $\pi_{\mathrm{T}}(m)$ represents the projection of m to T, it holds that m and $\pi_{\mathrm{T}}(m)$ have exactly the same set of factors under these relations. Moreover, if $\mathrm{T} = \Sigma$ it follows by definition that these are equivalent to their nonrelativized analogues. Therefore the only difference between a k-SL grammar and a corresponding k-TSL one is interpretation:

$$\mathbb{1}_G(m) \;=\; \mathscr{F}_k^{\triangleleft^{\mathrm{T}}}(m) \subseteq G.$$

We will see that this is indeed always the case, so from this point on characteristic functions will be given without the relation specified. Using $\triangleleft$ yields the local class, $\triangleleft^{\mathrm{T}}$ its relativization.

With this in mind, we turn to the language-theoretic characterization of the SL class: closure under substitution of suffixes (Rogers and Pullum, 2011, see also De Luca and Restivo, 1980). A stringset satisfies suffix substitution closure iff there is some k such that for any two strings $w_1 = u_1xv_1$ and $w_2 = u_2xv_2$ where $|x| \geqslant k-1$, if both w_1 and w_2 are in the set, then so is $w_3 = u_1xv_2$. Lambert and Rogers (2020) provide a similar characterization for the TSL class.

Definition 1 (Preprojective suffix substitution closure: PSSC)**.** A stringset L over the alphabet

Σ is closed under T-preprojective suffix substitution (T-PSSC) iff there is some natural number k such that for any two strings $w_1 = u_1x_1v_1$ and $w_2 = u_2x_2v_2$ where $\pi_{\mathrm{T}}(x_1) = \pi_{\mathrm{T}}(x_2)$ and $|\pi_{\mathrm{T}}(x_1)| \geqslant k - 1$, if both w_1 and w_2 are in L, then so is $w_3 = u_1x_1v_2$.

Notice that Σ-PSSC is equivalent to standard suffix substitution closure, as the strings are necessarily equal to their projections. On its own though, T-PSSC is not sufficient to characterize TSL. The other necessary condition is that symbols not in T be freely insertable and deletable, as previously discussed for the relativized relations.

Theorem 1. *A stringset L over an alphabet Σ is* TSL *iff there is some subset $\mathrm{T} \subseteq \Sigma$ such that symbols not in* T *are freely insertable and deletable and L is closed under* T-PSSC.

Proof. To prove that such a stringset is TSL, suppose there exists such a T. Then by T-PSSC, the projection of L to T is SL. Further, by insertion and deletion closure of non-T symbols we guarantee that w is in L iff its projection to T is. Together, these facts show that L is TSL.

To prove the reverse implication, suppose that L is TSL. Then it is k-TSL$^{\mathrm{T}}$ for some k and T. By definition of TSL, w is in L iff its projection to T is, and so symbols not in T are freely insertable and deletable. Because L is TSL$^{\mathrm{T}}$, its projection to T is SL and thus satisfies suffix substitution closure. Further since L is closed under insertion of symbols not in T, it is then also closed under T-PSSC. ■

In order to prove that a stringset is TSL, it suffices to provide a grammar. To prove that a stringset cannot be TSL, one can find some set of symbols that are not freely both insertable and deletable, then form strings from those symbols alone that violate PSSC. For instance, a constraint that forbids sequential (not necessarily adjacent) occurrences of "a . . b . . a" is not TSL because neither "a" nor "b" is freely insertable (so they are necessarily in T) and PSSC is violated by the following words:

$$\begin{array}{ll} \mathrm{a}\overbrace{\mathrm{b}\,..\,\mathrm{b}}^{k-1}\mathrm{b} & (\in) \\ \mathrm{b}\mathrm{b}\,..\,\mathrm{b}\mathrm{a} & (\in) \\ \hline \mathrm{a}\mathrm{b}\,..\,\mathrm{b}\mathrm{a} & (\notin). \end{array}$$

3.2.2 Complements

If an SL stringset contains all strings that satisfy a grammar G, the complement of this set is all strings that do not satisfy G. Since a model satisfies G iff all of its factors are in G, it follows that the model does not satisfy G iff it has at least one factor not in G. We can use the grammar of the complemented SL stringset as the grammar for a coSL stringset. The result is a collection of factors, at least one of which is required to appear in every word. The resulting characteristic function then is

$$\mathbb{1}_G(m) \;=\; G \cap \mathscr{F}_k(m) \neq \varnothing.$$

Definition 2 (Ideal containment: IC)**.** A stringset L is k-coSL iff L contains all and only those strings w that themselves contain at least one factor $f \in \mathscr{F}_k^{\triangleleft}(w)$ such that every string x that contains that factor is also in L:

$$\left\{x \colon f \in \mathscr{F}_k^{\triangleleft}(x)\right\} \subseteq L.$$

If there exists some k for which L is k-coSL, then L is coSL.

In IC, the factor f is the (not necessarily unique) factor that caused dissatisfaction of the corresponding SL grammar. Then in order to show that a stringset is not k-coSL, it suffices to find a (preferably small) word $w \in L$ and a set of words S such that $\mathscr{F}_k(w) \subseteq \mathscr{F}_k(S)$, yet no member of S is in L. For instance we can show that a constraint that bans occurrence of the substring “ab” is not coSL because IC is violated by the following:

$$\begin{aligned} w = {} & \mathrm{a}\,.\,.\,\mathrm{a} && (\in) \\ S = {} & \big\{\underbrace{\mathrm{a}\,.\,.\,\mathrm{a}}_{k-1}\mathrm{b}\underbrace{\mathrm{a}\,.\,.\,\mathrm{a}}_{k-1}\big\} && (\notin). \end{aligned}$$

The factors of w are $\{\rtimes\mathrm{a}^i, \mathrm{a}^i\ltimes : 0 \leqslant i < k\}$. Each of these factors occurs in the single word in S, and so the presence of any given factor is not sufficient to guarantee acceptance. This extends trivially to **preprojective ideal containment**.

Definition 3 (Preprojective ideal containment: PIC)**.** A stringset L is k-coTSL iff there exists some $\mathrm{T} \subseteq \Sigma$ such that L contains all and only those strings w that themselves contain at least one factor $f \in \mathscr{F}_k^{\lhd\mathrm{T}}(w)$ such that every string x that contains that factor is also in L:

$$\big\{x : f \in \mathscr{F}_k^{\lhd\mathrm{T}}(x)\big\}.$$

And L is coTSL iff it is k-coTSL for some k.

Since the factors of w under $\lhd^{\mathrm{T}}$ are the same as those of its T-projection under $\lhd$, this is equivalent in every way to stating that L is k-coTSL$^{\mathrm{T}}$ iff its T-projection is k-coSL and it is closed under insertion and deletion of symbols not in T. Further the order of complementation and relativization is immaterial, as will become clear in section 3.3.

3.2.3 Local testability

Much as the SL stringsets consist of words containing only permitted factors, the locally testable (LT) ones consist of words whose set of factors is permitted (McNaughton and Papert, 1971). This allows a mechanism to reject "ab" even in the case of accepting "abab", whose 2-factors are $\{\rtimes\mathrm{a}, \mathrm{ab}, \mathrm{b}\ltimes\}$ and $\{\rtimes\mathrm{a}, \mathrm{ab}, \mathrm{ba}, \mathrm{b}\ltimes\}$, respectively. Due to their subset relationship, a mechanism capable only of 2-SL distinctions could not do this, though each of the three other possible combinations of acceptance and rejection of these two strings is possible under 2-SL. For testable stringsets, a grammar is a set of permitted sets of factors, and its characteristic function is

$$\mathbb{1}_G(m) \;=\; \mathscr{F}_k(m) \in G.$$

Rogers and Pullum (2011) state that a stringset is LT iff it is closed under **local test invariance**, where given two strings w_1 and w_2 such that $\mathscr{F}_k^{\lhd}(w_1) = \mathscr{F}_k^{\lhd}(w_2)$, the first is in the set iff the second is as well. This extends trivially to **preprojective local test invariance**.

Definition 4 (Preprojective local test invariance: PLTI)**.** A stringset L is TLT iff there exists some $\mathrm{T} \subseteq \Sigma$ and some k such that given two strings w_1 and w_2 such that $\mathscr{F}_k^{\lhd^{\mathrm{T}}}(w_1) = \mathscr{F}_k^{\lhd^{\mathrm{T}}}(w_2)$, the first is in L iff the second is as well.

By the same reasoning employed in discussion of coTSL, this is equivalent in every way to stating that L is k-TLT$^{\mathrm{T}}$ iff its T-projection is k-LT and it is closed under insertion and deletion of symbols not in T.

3.2.4 Threshold testability

The LT class is characterized by sets of factors. But a set is merely a structure that describes each possible element by a Boolean value, whether or not that element is included. One might consider a natural extension of this structure which saturates its count of occurrences not at 1 but at some arbitrary value t. This is exactly what Beauquier and Pin did when defining the **locally threshold testable** (LTT) stringsets in 1989. We denote this generalized structure by

$$\mathscr{F}_{k,t}(m) \triangleq \lbag [\![x]\!]_m \colon x \in \mathscr{W}_k \rbag_t .$$

For threshold testable stringsets, a grammar is a set of permitted multisets whose characteristic function is

$$\mathbb{1}_G(m) \;=\; \mathscr{F}_{k,t}(m) \in G.$$

The characterization of LTT of course is **local threshold test invariance**, where given two strings w_1 and w_2 such that $\mathscr{F}^{\triangleleft}_{k,t}(w_1) = \mathscr{F}^{\triangleleft}_{k,t}(w_2)$, the first is in the set iff the second is as well. This extends trivially to **preprojective local threshold test invariance**.

Definition 5 (Preprojective local threshold test invariance: PLTTI)**.** A stringset L is TLTT iff there exists some $\mathrm{T} \subseteq \Sigma$ and some k and t such that given two strings w_1 and w_2 such that $\mathscr{F}^{\triangleleft^{\mathrm{T}}}_{k,t}(w_1) = \mathscr{F}^{\triangleleft^{\mathrm{T}}}_{k,t}(w_2)$, the first is in L iff the second is as well.

By the same reasoning employed in discussion of coTSL and TLT, this is equivalent in every way to stating that L is k, t-TLTT$^{\mathrm{T}}$ iff its T-projection is k, t-LTT and it is closed under insertion and deletion of symbols not in T.

Under the definitions of windows and factors used in this text, this actually counts prefixes and suffixes of length less than k more than once, since several windows might correspond to the same factor. Importantly, for fixed k the count is consistent for prefixes (suffixes) of a given length, and since there is at most one length-n prefix (suffix) in any valid word model, this overcounting does not affect the possible distinctions.

3.2.5 Piecewise relativizations

Using general precedence instead of successor in the definition of the SL class yields the strictly piecewise (SP) class. Characterized by Rogers et al. (2010) as those stringsets closed under deletion (see also Haines, 1969 for an earlier treatment of stringsets closed under deletion), we can show that a stringset is TSP iff it is SP. In fact, unlike reduction, relativization provides neither more nor less expressive power in precedence-based models.

By definition $<^{\Sigma}$ is equivalent to $<$, thus all nonrelativized stringsets are also trivially relativized ones. More interesting is the reverse. Recall from Figure 3.1 that the relativization of the $<$ relation is in fact merely a restriction, and so $\mathscr{F}_k^{<^{\mathrm{T}}}(m) \subseteq \mathscr{F}_k^{<}(m)$. If a factor occurs on the restriction, then it also occurs in the nonrestricted model. Therefore a full piecewise model must be at least as powerful as its relativization, but since T can be equal to Σ they are in fact equivalent.

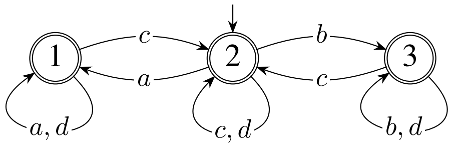
For this same reason, when the factor width is fixed at $k = 1$ all of the projectively relativized classes are equivalent to their nonrelativized analogues.

3.3 Automata

In this section we discuss characterizations of the relativized classes and constructions of their constituent stringsets in terms of deterministic finite-state automata (DFAs).

Recall that a DFA is a directed graph that represents a machine that computes the well-formedness of a string with respect to some regular stringset, and is represented by a five-tuple $\langle \Sigma, Q, \delta, q_0, F \rangle$, Σ an alphabet, where Q is a set of states, $\delta \colon \Sigma \times Q \to Q$ a transition function which represents edges in the graph, $q_0 \in Q$ an initial state, and $F \subseteq Q$ a set of accepting states. A DFA is **complete** iff δ is total. A word w is accepted by the DFA iff there is some path $q_0 \to \cdots \to q_f$ for some $q_f \in F$ (an **accepting path** from q_0) whose labels spell out w.

Let $\sim$ represent the equivalence relation over states in Q under which $q_1 \sim q_2$ iff for all $u \in \Sigma^*$ there is an accepting path from q_1 labeled u whenever such a path exists from q_2 and vice-versa. This is **Nerode equivalence**. By the Myhill-Nerode theorem (Nerode, 1958), one can construct a **minimal** DFA from a given one by replacing each state in Q by its Nerode-equivalence class, the element of $Q/{\sim}$ that contains it. A minimal DFA might have a unique **nonaccepting sink**, a state q from which there are no accepting paths for any string. A **canonical** DFA is one that is minimal and has had its nonaccepting sink (if any) removed.



$$\complement\mathrm{T} = \bigcap\big\{\{a,d\},\{c,d\},\{b,d\}\big\} = \{d\}$$

Figure 3.3: A canonical DFA for which d is a nonsalient symbol.

3.3.1 Characterizations

Every relativized class discussed in this text is closed under insertion and deletion of symbols not in T. In other words, such symbols provide no information regarding the well-formedness of a word. It follows then that from a given state q, the state reached by following an edge labeled by such a symbol must be in the same Nerode-equivalence class as q itself. Thus in a canonical DFA, the nonsalient symbols are exactly those that form self-loops on all states simultaneously.

$$\complement\mathrm{T} = \bigcap_{q\in Q}\{\sigma\in\Sigma\colon \delta_\sigma(q) = q\}.$$

In other words, these are exactly the symbols σ such that the set of fixed points of δ_σ is the entirety of Q. Figure 3.3 shows a canonical DFA that represents a TSL stringset in which d is not a salient symbol. The fixed points for a, b, c, and d are $\{1\}$, $\{3\}$, $\{2\}$, and $\{1,2,3\}$, respectively.

Given a canonical automaton for a language L, the automaton for its T-projection is formed by restricting δ to $\Sigma\setminus\complement T$. As discussed previously, with this choice of T a stringset is in the relativized variant of a given class iff its T-projection is in that class. The T found this way is in fact the smallest set for which this holds, though some stringsets might permit several possible values for T. For example, the stringset Σ^* is TSL^P for every $P\subseteq\Sigma$.

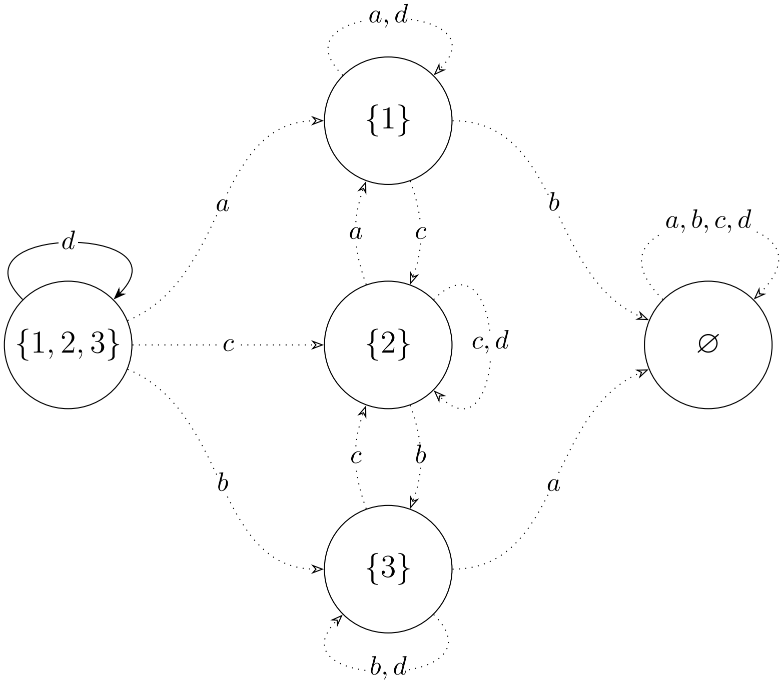


Figure 3.4: The powerset graph that corresponds to the DFA of Figure 3.3, with the cycle marked that proves this stringset is not SL. Notice that if d is removed, there is no such cycle.

Once the projection has been found, any of the numerous existing methods for determining class membership can be used. Most of these tests are based on the algebraic interpretation of the syntactic semigroup corresponding to the automaton, which will be discussed further in section 3.5. However, for the SL class we can use a result of Edlefsen et al. (2008, see also Caron, 1998). Given a canonical DFA $A = \langle \delta, q_0, F \rangle$, construct its **powerset graph** by defining $\delta^{\mathscr{P}} \colon \Sigma \times \mathscr{P}(Q) \to \mathscr{P}(Q)$:

$$\delta_\sigma^{\mathscr{P}}(S) \triangleq \{\delta_\sigma(s) \colon s \in S\}.$$

The stringset represented by A is SL iff the graph formed by $\delta^{\mathscr{P}}$ contains no cycles that iterate a node whose label contains two or more elements. A powerset graph of a non-SL stringset is shown in Figure 3.4 with the offending path marked.

Edlefsen et al. (2008) also provide a more efficient algorithm in terms of pairs of states. However, the powerset graph construction also allows extraction of a grammar for the target stringset from the automaton itself (Rogers and Lambert, 2019b). Since a stringset is coSL iff its complement is SL, this serves as an automata-theoretic method to decide membership in that class as well. Thus by projecting an automaton to the appropriate tier alphabet, not only can we show that the target stringset is TSL or coTSL, but we can also obtain a canonical grammar for this stringset if it is.

3.3.2 Constructions

The projection mechanism of section 3.3.1 is invertible. Given a DFA representing a stringset L whose alphabet is some $\mathrm{T} \subseteq \Sigma$, the preprojection of L to Σ is given by adding self-edges on each symbol $\sigma \in \Sigma \setminus \mathrm{T}$ to every state. If the subregular class of L is known, then the result lies in the corresponding relativized class.

However, the purpose of these projectively relativized classes is to represent those stringsets in which only certain symbols are salient. It would be meaningful then to employ **alphabet-agnostic automata** (AAA). Such automata test for the occurrence of a factor within a target string of unknown or unspecified alphabet by augmenting the set of symbols relevant to the factor with a **wildcard** symbol $\textcircled{?}$, much like Beesley and Karttunen (2003). For our purposes, we consider a wildcard that matches all and only those symbols not already listed in the alphabet, like the @ of Hulden (2009). The empty language is represented by a single state which is non-accepting, bearing a self-loop labeled $\textcircled{?}$, which is the only member of the alphabet. The universal acceptor is identical, except its single state is accepting.

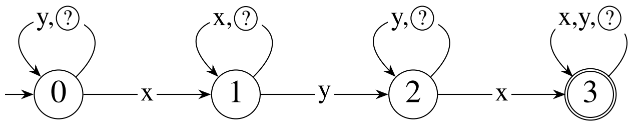


Figure 3.5: Canonical AAA for the factor “xyx” under $<$.

Factors under $<$ (piecewise factors) are the simplest to construct. Given a factor $f = \sigma_1 \dots \sigma_n$ under this relation, the AAA is defined in essentially the same way that Rogers et al. (2010) form a DFA:

$$
\begin{aligned}
Q &= \{0, \dots, n\} \\
\Sigma &= \{\sigma_1, \dots, \sigma_n, \textcircled{?}\} \\
q_0 &= 0 \\
F &= \{n\} \\
\delta(\sigma, q) &= \begin{cases} q + 1 & \text{if } q < n \text{ and } \sigma = \sigma_{q+1}, \\ q & \text{otherwise.} \end{cases}
\end{aligned}
$$

An example is shown in Figure 3.5. Note that since $\textcircled{?}$ is by definition never included in the factor, edges on this symbol are always self-loops. In other words, this construction provides a clear picture as to why SP (and extensions thereof) and TSP (and extensions) should be identical.

For factors under $\lhd$ (local factors), any of the common approaches to substring-matching suffice, including that of Knuth et al. (1977). However, a naïve method shown here demonstrates some properties of AAA. Fully-anchored factors of the form $f = \rtimes\sigma_1 \dots \sigma_n\ltimes$ look like piecewise factors,

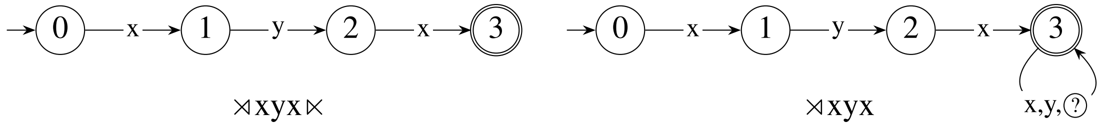


Figure 3.6: Canonical automata for head-anchored factors.

except edges to a non-accepting sink $\perp$ replace the self-loops:

$$Q = \{\perp, 0, \ldots, n\}$$

$$\delta(\sigma, q) = \begin{cases} q+1 & \text{if } q < n \text{ and } \sigma = \sigma_{q+1} \\ \perp & \text{otherwise.} \end{cases}$$

For head-anchored but not tail-anchored factors, the difference is that $\delta(\sigma, n) = n$ rather than $\perp$. Figure 3.6 shows a canonical (trimmed) AAA constructed for each of “⋊xyx⋉” and “⋊xyx”.

These automata can then be concatenated after a universal acceptor to represent tail-anchored and free factors, as in Figure 3.7, but this concatenation requires an extra step compared to standard DFA operations: the two inputs to any binary operation must be made **compatible**. Two AAA are compatible iff they have the same alphabet. Since ?⃝ represents any symbol not already in the alphabet, a symbol σ is added by placing new edges labeled by σ in parallel with any existing ?⃝ edges. Thus to make two AAA compatible, their alphabets should be extended in this way to the union of their individual alphabets. Two compatible automata can be combined using standard DFA operations, treating ?⃝ as just another symbol.

To fix the alphabet of an AAA to a specific set Σ, simply extend it as necessary and then remove any ?⃝ edges. Relativizing an automaton over some tier alphabet T is a process of fixing its alphabet to

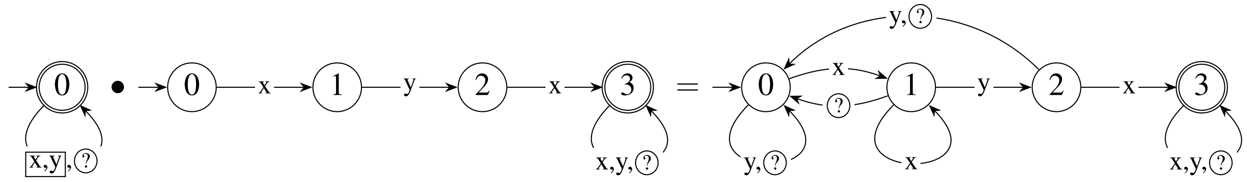


Figure 3.7: Canonical AAA for the free factor “xyx” under $\lhd$ constructed via concatenation. The boxed $\boxed{\text{x,y}}$ in the first operand are symbols that needed to be added for compatibility.

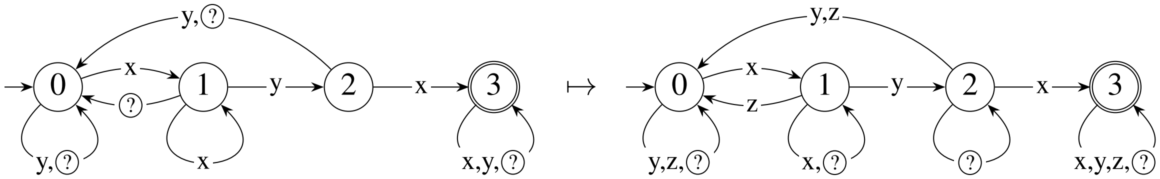


Figure 3.8: Relativizing “xyx” to $\mathrm{T} = \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$: Add ‘z’ in parallel to any existing ? edges, remove those ? edges, then add new ? edges as self-loops everywhere.

T, then adding ? edges back in as self-loops on every state. For example, Figure 3.8 shows how this process modifies the factor of Figure 3.7 to consider only ‘x’, ‘y’, and ‘z’ salient.

3.4 Closure Properties

In this section we constructively prove some closure properties of relativized classes via their automata-theoretic characterizations, and use the language-theoretic ones to provide counter-examples to other closure properties.

3.4.1 Products

The intersections and unions of automata are both formed from **the product construction**. Given two automata $A = \langle \delta, q_0, F \rangle$ and $A' = \langle \delta', q_0', F' \rangle$, one constructs the product

$$A \otimes_{op} A' \triangleq \big\langle \delta^{\otimes}, \langle q_0, q_0' \rangle, F^{op} \big\rangle,$$

where the new transition function is defined pointwise:

$$\delta_\sigma^{\otimes}\bigl(\langle q, q'\rangle\bigr) \triangleq \bigl\langle \delta_\sigma(q), \delta'_\sigma(q')\bigr\rangle.$$

The set of accepting states is

$$F^{op} \triangleq \bigl\{\langle q, q'\rangle\colon q \in F \;\, op \;\, q' \in F'\bigr\},$$

where op is “and” for intersection or “or” for union. If σ labels a self-loop on every state of both operands, then this product construction guarantees that the same will hold in the result. By construction then, if $\complement\mathrm{T}$ is the set of nonsalient symbols for A and $\complement\mathrm{T}'$ for A', then $\complement\mathrm{T}^{\otimes} \subseteq \complement\mathrm{T} \cap \complement\mathrm{T}'$ is a set for this product.[1] Notably, if A and A' represent stringsets in a relativized class where $\mathrm{T} = \mathrm{T}'$ and the underlying class is closed under union (intersection), the result of the union (intersection) of A and A' remains in the same relativized class with the same set of salient symbols.

Theorem 2. *The* TLT$^{\mathrm{T}}$ *and* TLTT$^{\mathrm{T}}$ *classes for fixed* T *are closed under both union and intersection.*

Proof. Because LT and LTT are each closed under both union and intersection, the product construction guarantees that TLT$^{\mathrm{T}}$ and TLTT$^{\mathrm{T}}$ for fixed T are as well. ■

For the same reasons, the same holds for TSL$^{\mathrm{T}}$ under intersection and coTSL$^{\mathrm{T}}$ under union.

Note that these closures rely on equality of the sets of salient symbols. Consider the TSL stringset whose projection to $\{\mathrm{a}, \mathrm{b}\}$ contains no “ab” factor and the TSL stringset whose projection to $\{\mathrm{a}, \mathrm{c}\}$

[1] Since the result is not necessarily canonical, a larger $\complement\mathrm{T}^{\otimes}$ (thus a smaller T) may also exist.

contains no “aca” factor. In the intersection, none of “a”, “b”, or “c” is freely insertable, so each must be salient no matter what Σ is. Then even though the two strings

$$\begin{array}{cc} \overbrace{\mathrm{c}\,.\,.\,\mathrm{c}}^{k}\,\mathrm{b}\,\overbrace{\mathrm{c}\,.\,.\,\mathrm{c}}^{k}\,\mathrm{a}\,\overbrace{\mathrm{c}\,.\,.\,\mathrm{c}}^{k} & (\in) \\ \mathrm{c}\,.\,.\,\mathrm{c}\,\mathrm{a}\,\mathrm{c}\,.\,.\,\mathrm{c}\,\mathrm{b}\,\mathrm{c}\,.\,.\,\mathrm{c} & (\notin) \end{array}$$

have exactly the same k-factors and exactly the same counts for each, the first is in the intersection while the second is not. This is a violation of PLTTI (see page 32). Thus the intersection of these two stringsets is not even TLTT, and so by containment it cannot be TLT or TSL. For a more direct proof that this intersection is not TSL, consider the following violation of PSSC:

$$\begin{array}{cc} \mathrm{ac}\,\overbrace{\mathrm{c}\,.\,.\,\mathrm{c}}^{k-1}\,\mathrm{ca} & (\in) \\ \mathrm{b}\,\mathrm{c}\,.\,.\,\mathrm{c}\,\mathrm{b} & (\in) \\ \hline \mathrm{ac}\,\mathrm{c}\,.\,.\,\mathrm{c}\,\mathrm{b} & (\notin). \end{array}$$

In this example, the result of PSSC is a string whose projection to $\{\mathrm{a}, \mathrm{b}\}$ contains an ab factor, which should be forbidden.

3.4.2 Complements of automata

To find the complement of a complete minimal DFA, simply invert the notion of acceptance. That is, map $\langle\delta, q_0, F\rangle$ to $\langle\delta, q_0, Q \setminus F\rangle$.

Theorem 3. *A regular stringset L and its complement can be defined by expressions over the same tier of salient symbols as one another.*

Proof. Because L is regular, it can be represented as a complete minimal DFA, and this DFA will be associated with some set of salient symbols. The complement operation does not affect the transition function δ, so the set of self-loops in the result is exactly the same as that in the input. It follows then that the complement of L has the same set of salient symbols as L itself. ■

Then if the underlying class is closed under complementation (as is the case for LT and LTT) the corresponding relativized variant is so closed as well. Moreover, since a relativization is formed by merely adding self-loops everywhere, the order of relativization and complementation is immaterial. The two operations cannot interfere with one another.

3.4.3 Some non-closures

Having shown that, for fixed T, TLT$^{\mathrm{T}}$ and TLTT$^{\mathrm{T}}$ are closed under all Boolean operations and TSL$^{\mathrm{T}}$ and coTSL$^{\mathrm{T}}$ are closed under intersection and union, respectively, we now show that TSL$^{\mathrm{T}}$ is not closed under union or complement, and that coTSL$^{\mathrm{T}}$ is not closed under intersection or complement.

Consider two TSL$^{\{\mathrm{a},\mathrm{b}\}}$ stringsets: one which bans the occurrence of “ab” on the projection to $\{\mathrm{a}, \mathrm{b}\}$, and another which bans the occurrence of “ba” on this projection. The union of these two stringsets allows the occurrence of either “ab” or “ba”, but not both. Then the following is a violation of PSSC:

$$\begin{array}{rl} \mathrm{ab}\,\overbrace{\mathrm{b}\,.\,.\,\mathrm{b}}^{k-1} & (\in) \\ \mathrm{b}\,.\,.\,\mathrm{b}\,\mathrm{ba} & (\in) \\ \hline \mathrm{ab}\,\mathrm{b}\,.\,.\,\mathrm{b}\,\mathrm{ba} & (\notin). \end{array}$$

The complement of the first of these, that “ab” must occur on the projection to $\{\mathrm{a}, \mathrm{b}\}$, is also not TSL. The following is a violation of PSSC:

$$\begin{array}{ll} \overbrace{\mathrm{a}\,..\,\mathrm{a}}^{k-1}\,\mathrm{ab} & (\in)\\ \mathrm{ab}\,\mathrm{a}\,..\,\mathrm{a} & (\in)\\ \hline \mathrm{a}\,..\,\mathrm{a} & (\notin). \end{array}$$

Now consider two $\text{coTSL}^{\{\mathrm{a},\mathrm{b}\}}$ stringsets, one which requires that some “ab” occurs on the projection to $\{\mathrm{a}, \mathrm{b}\}$, and another that requires that some “ba” occurs on this projection. Their intersection (requiring both to occur) is not coTSL, as the following violates PIC:

$$\begin{array}{rll} w = & \mathrm{a}\,\overbrace{\mathrm{b}\,..\,\mathrm{b}}^{k-1}\,\mathrm{a} & (\in)\\ S = & \big\{\mathrm{a}\,\mathrm{b}\,..\,\mathrm{b}, & \\ & \qquad \mathrm{b}\,..\,\mathrm{b}\,\mathrm{a}\big\} & (\text{each } \notin), \end{array}$$

since the collective factorset of S is a superset of the factors of w. And of course, the complement of the first of these stringsets, banning occurrences of “ab” on the $\{\mathrm{a}, \mathrm{b}\}$-projection, is also not coTSL, because, as shown in section 3.2.2, the stringset of the projection is not coSL.

From this we have shown that TLTT^{T} and TLT^{T} are closed under all Boolean operations, while TSL^{T} and coTSL^{T} are closed only under intersection and union, respectively. We have also shown that intersection and union closures hold only when both stringsets have the same set of salient symbols.

3.5 Algebra

Many of the algorithms that decide whether a given DFA represents a stringset from a particular class actually make use of the **syntactic semigroup** associated with the DFA. Given a complete minimal DFA $A = \langle \delta, q_0, F \rangle$, recall that $\delta \colon \Sigma \times Q \to Q$ can be viewed as $\hat{\delta} \colon \Sigma \to (Q \to Q)$ and define $\gamma \colon \Sigma^* \to (Q \to Q)$ as follows:

$$\gamma(w) \triangleq \begin{cases} \gamma(v) \circ \hat{\delta}(\sigma) & \text{if } w = \sigma v \text{ for some } \sigma \in \Sigma \text{ and } v \in \Sigma^* \\ \text{id} & \text{otherwise.} \end{cases}$$

Here, 'id' refers to the identity function. The syntactic semigroup is then the semigroup under flipped composition of the following functions:

$$S(A) \triangleq \{\gamma(w) \colon w \in \Sigma^+\}.$$

Then if $a = \gamma(u)$ and $b = \gamma(v)$, we have $ab = b \circ a = \gamma(v) \circ \gamma(u) = \gamma(uv)$. Since $\gamma(u)\gamma(v) = \gamma(uv)$, the semigroup operation is a homomorphism.

The star-free stringsets are those whose syntactic semigroup is finite and has no nontrivial subgroups (Pin, 1997).

Recall from section 3.3 that the set of nonsalient symbols, $\complement\mathrm{T}$, is the set of symbols that form self-loops on every state of a DFA. Translated to the algebraic domain, these are all and only those symbols σ for which $\gamma(\sigma) = \mathrm{id}$. Sometimes we denote id by 1. If $1 \in S$, then we also say S is a monoid.

Lemma 1. *If S is the syntactic semigroup of a star-free stringset and $a, b \in S$ are elements such that $ab = 1$, then $a = b = 1$.*

Proof. Suppose a and b are elements of S such that $ab = 1$, and that S is the syntactic semigroup of a star-free stringset. Then $1 = ab = (a(ab))b$, and by continuing this process we see that $a^n b^n = 1$ for all n. Schützenberger (1965) proves that because S is star-free, there exists some m such that $a^m = a^{m+1}$. Then we have $1 = a^m b^m = a^{m+1} b^m = a a^m b^m = a1 = a$, and by a similar argument we see that $b = 1$ as well. Therefore $a = b = 1$. ■

This means that if S corresponds to a star-free stringset, then every w such that $\gamma(w) = \mathrm{id}$ is composed entirely of symbols from $\complement\mathrm{T}$. We now define the **projected subsemigroup** as

$$X(A) \triangleq \bigl\{\gamma(w) \colon w \in \mathrm{T}^{+}\bigr\}.$$

Theorem 4. *If S corresponds to a star-free stringset, then the projected subsemigroup X of S is equal to $S \setminus \{1\}$.*

Proof. Suppose $s \in S$. Then $s = \gamma(w)$ for some $w \in \Sigma^{*}$, and by definition if w is the string $\sigma_1 \ldots \sigma_n$ then $s = \gamma(\sigma_n) \circ \cdots \circ \gamma(\sigma_1)$.

Suppose $s \neq 1$. Because whenever $\sigma \in \complement\mathrm{T}$ it holds that $\gamma(\sigma) = 1$, it follows that $\gamma(w) = \gamma\bigl(\pi_{\mathrm{T}}(w)\bigr)$. By definition, if $|\pi_{\mathrm{T}}(w)| \neq 0$ then $s \in X$ as well. If the length of this projection were 0, then $s = \gamma\bigl(\pi_{\mathrm{T}}(w)\bigr)$ would be 1, and since by assumption $s \neq 1$, this cannot be. Therefore, $s \in X$.

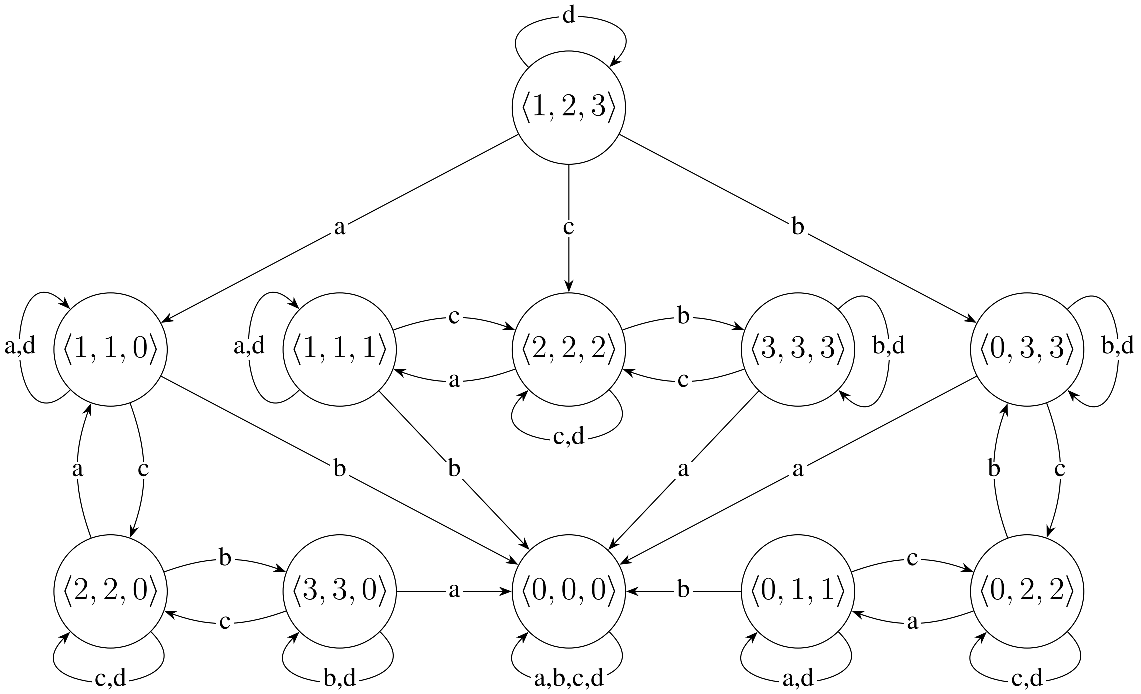


Figure 3.9: The syntactic semigroup corresponding to the DFA of Figure 3.3. Each function is represented by the image of $\langle 1, 2, 3\rangle$. (Each tuple additionally has a fourth entry that is always labeled 0 and is omitted here for brevity.) The corresponding projected subsemigroup lacks the identity as well as every edge labeled d.

On the other hand, suppose $s = 1$. Then by Lemma 1, we have $\{\sigma_1, \ldots, \sigma_n\} \subseteq \complement\mathrm{T}$. Then $w = \sigma_1 \ldots \sigma_n$ is not in T^* and by definition is excluded from X.

Thus if S is star-free, X contains all and only the nonidentity elements of S. ■

Because S and X are finite and generated by Σ or T, respectively, we can visualize them as edge-labeled directed graphs just like a DFA. If A is the DFA shown in Figure 3.3 on page 35 augmented with a nonaccepting sink labeled 0, then S is the semigroup shown in Figure 3.9.

3.5.1 Strictly local stringsets and their complements

The SL and coSL classes (and their relativizations) cannot be characterized algebraically solely on the basis of their syntactic semigroups. Because the syntactic semigroup is generated exclusively from the transition function δ of a DFA, an automaton and its complement are associated with the same semigroup. This fact has been noted previously by McNaughton (1974). However, if information describing whether each state is accepting or rejecting (state parity) is retained then an algebraic characterization is possible.

De Luca and Restivo (1980) provide a characterization based on Schützenberger's (1975) concept of a **constant**. Let A be a subset of some semigroup S, and let $c \in S$. Then c is a constant for A if for all $p, q, r, s \in S \cup 1$ it holds that whenever $pcq \in A$ and $rcs \in A$ it follows that $pcs \in A$. If S is freely generated from some finite alphabet, De Luca and Restivo prove that A is SL iff all sufficiently long words of S are constants for A. This is equivalent to the suffix-substitution closure discussed in section 3.2.1. Of course if all such words are instead constants for the complement of A, then A is coSL rather than SL.

In this case, A is the stringset being tested, the language generated by the syntactic semigroup under observation. If instead these properties hold for the projected subsemigroup, then the stringset is TSL or coTSL, respectively. But again, no algorithm can distinguish a stringset and its complement given only an unadorned syntactic semigroup.

3.5.2 Locally testable stringsets

The characterization for LT and therefore TLT is due to Brzozowski and Simon (1973). First, note that an element x of a semigroup S is called **idempotent** iff $x = xx$. An **idempotent semigroup** is a semigroup such that all of its elements are idempotent. Given the syntactic semigroup S of a stringset L, Brzozowski and Simon proved that L is LT iff for all idempotent elements e of S it holds that the subsemigroup eSe is a commutative idempotent monoid. Note that the monoid requirement is immaterial, as eSe will always be a semigroup with identity. Namely, for any $s \in S$, we see that $eee \cdot ese = (ee)(ee)se = eese = ese$ and similarly $ese \cdot eee = ese$, so eee (which is itself simply e) is the identity.

Theorem 5. *A stringset L is* TLT *iff for all idempotent elements e of X, the projected subsemigroup of its syntactic semigroup, it is the case that eXe is a commutative idempotent semigroup.*

This holds because the projected subsemigroup is equivalent to the syntactic semigroup of the projection, and a stringset is TLT iff its projection is LT by definition.

3.5.3 Locally threshold testable stringsets

The characterization for LTT and therefore TLTT is due to Beauquier and Pin (1989). They define a variety (collection) of semigroups $\mathbf{V}$ to contain all and only those aperiodic semigroups S such that if e and f are idempotent, $p = es_1f$, $q = fs_2e$, and $r = es_3f$, it holds that $pqr = rqp$. They then prove that a language L is LTT iff its syntactic semigroup S is a member of $\mathbf{V}$. By expansion, this means that for all idempotents e, f of S and for all $s_1, s_2, s_3 \in S$, it holds that $es_1fs_2es_3f = es_3fs_2es_1f$.

The aperiodicity requirement is meaningful, as there do exist stringsets that satisfy the latter requirement without even being star-free. Consider the set of strings of even length over a unary alphabet, $(aa)^*$. Its syntactic semigroup consists of two elements, 1 and a, such that $aa = 1$. It necessarily holds that $es_1fs_2es_3f = es_3fs_2es_1f$ under any instantiation of these variables, because this is a commutative monoid.

Theorem 6. *Let L be a stringset and X be the projected subsemigroup of its syntactic semigroup. Then L is* TLTT *iff for all idempotent elements e, f of X, and for all s_1, s_2, s_3 in X, it holds that $es_1fs_2es_3f = es_3fs_2es_1f$.*

This holds because the projected subsemigroup is equivalent to the syntactic semigroup of the projection, and a stringset is TLTT iff its projection is LTT by definition.

3.6 Conclusions

We introduced several new classes to the piecewise-local subregular hierarchy building from the model-theoretic characterization of TSL by Lambert and Rogers (2020) and noting that projective relativization does not affect the expressivity of classes based on general precedence. We provided model-, language-, and automata-theoretic as well as algebraic characterizations for each new class. This establishes a clearer notion of relativized adjacency based on the salience of individual symbols, and informs linguistic theory as it pertains to long-distance dependencies in phonology.

The topic of learning is not covered in this work. But we would be remiss to ignore that the unifying concept of relativized adjacency provides a straightforward mechanism to extend known learning results for the TSL class to TLT and TLTT (see McMullin, 2016; Jardine and Heinz, 2016; Jardine and McMullin, 2017).

A Haskell library[2] containing the automata-theoretic and algebraic decision algorithms has been created. With this one can construct regular and subregular stringsets from factors, or import OpenFST-format automata, and determine which, if any, subregular classes the result occupies. The factor-based constructors make use of the alphabet-agnostic automata described in section 3.3.2.

The class formed by intersecting multiple TSL constraints over different tier alphabets, MTSL (De Santo and Graf, 2019), is not explored here. Under this model-theoretic view, the relativized successor relation $\triangleleft^{\mathrm{T}}$ is parameterized by the alphabet over which it is relativized, so different instantiations of this relation are as different as the $\triangleleft$ and $<$ relations. With this in mind, future work in understanding these interactions will shed light on the strictly piecewise-local class discussed in Rogers and Lambert (2019a), and vice-versa. Moreover, Aksënova and Deshmukh (2018) discuss attested interactions of multiple tiers; one might ask which kinds of combinations (if any) allow intersection-closure to be maintained. Counterexamples exist for each type of interaction (disjoint, overlapping, and subset configurations), but perhaps some smaller portion of the constraint space may combine more readily.

The characterizations provided here of the relativized variants of the subregular classes rely heavily on the fact that $\triangleleft^{\mathrm{T}}$ and $<^{\mathrm{T}}$ are the results of a projective relativization. Another direction for future work then would be accounting for arbitrary nonprojective relations, which would improve our understanding of De Santo and Graf's structure-sensitive TSL class and its (threshold) testable extensions, or Graf's (2017) domain- and interval-based strictly piecewise classes.

[2] Software available at `https://github.com/vvulpes0/Language-Toolkit-2`

Finally, some of the subregular classes have been explored in application to trees (Rogers, 1997), and some extended to transducers (Chandlee, 2014; Ji and Heinz, 2020). It would be interesting to see how relativization applies to these cases as well.

Chapter 4: Learning Tier-Based Strictly Local Languages

The strictly local (SL) class known from McNaughton and Papert (1971) cannot account for long-distance dependencies. Meanwhile the strictly piecewise (SP) class of Rogers et al. (2010, see also Haines, 1969) can account for long-distance dependencies but cannot handle local constraints. In 2011, Heinz et al. presented a new tier-based strictly local (TSL) class of stringsets, generalizing SL to account for both local phenomena and certain types of long-distance phenomena by relativizing adjacency over a subset of the alphabet. Much like the traditional SL and SP classes, TSL is parameterized by the width k of the factors to which a learner or acceptor attends. But there is another parameter. If the subset T of the alphabet over which adjacency is relativized is known *a priori*, then the input data can be preprocessed by deleting unnecessary symbols and the grammar inferred by any of a number of strictly local learning algorithms, such as that of Garcia et al. (1990) or that of Heinz (2010b). But learning T from data alone has proven difficult, especially in a bounded-memory setting.

Since the introduction of the TSL class, it has been characterized model-, language-, and automata-theoretically by Lambert and Rogers (2020), extended to an even richer class by De Santo and Graf (2019), and shown to be batch-learnable in the style of Gold (1967) by Jardine and Heinz (2016) for k of 2 and by Jardine and McMullin (2017) for arbitrary k. But despite all this progress, no general online learning algorithm has yet been produced, leaving the TSL class behind in an area where several other subregular classes flourish.

That changes now. Here we discuss an online learning algorithm in the style of Heinz (2010b) that essentially combines the approaches taken in the earlier batch-learning algorithms with a novel representation of the TSL grammar to accommodate online learning. This development removes one argument against TSL descriptions of phonological patterns, namely that human learners likely do not operate in batch. (Batch learning requires perfect awareness of all prior input, which is implausible at best for human learners.)

Section 4.1 discusses background material. Then section 4.2 summarizes prior literature on discovering the set of salient symbols and provides space- and time-complexity analyses of the algorithms. Section 4.3 introduces a structure that can be gathered while determining salience, which provides sufficient information to recover the grammar of forbidden factors while avoiding unbounded data storage. Then section 4.4 demonstrates a pointwise application of the string extension learning algorithm of Heinz (2010b) to this problem, and introduces a simplification that allows for reinterpretation of an SL grammar as a TSL one.

4.1 Preliminaries

This section contains background material fundamental to this work. Section 4.1.1 provides a brief overview of the SL and TSL classes, with examples. Section 4.1.2 discusses the learning framework in use, and section 4.1.3 details the family of learning algorithms that includes our result.

4.1.1 (Tier-Based) Strict Locality

In general, a **factor** is some substructure of a word that is connected in some sense. For the SL languages as defined by McNaughton and Papert (1971), these substructures are simply adjacent sequences of symbols. For example, there are seven factors in the string “abc”: “”, “a”, “b”, “c”, “ab”, “bc”, and “abc” itself. Notably, “ac” is not a factor under this interpretation. The size of a factor is the length of the sequence. An SL language is characterized by a set of forbidden factors, containing all and only those strings in which no forbidden factor occurs. If the largest factor in the set of forbidden factors is of size k, then the language is k-SL.

One example of an SL language over the alphabet $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$ is that in which the forbidden factors are “aa” and “bb”. This requires that “a” and “b” alternate within blocks of these two symbols, so “abaca” and “abada” are valid words but “**aa**ca” is not.

One can prove that a language is not k-SL by using what is known as Suffix Substitution Closure: finding two valid words $w = w_p x w_s$ and $v = v_p x v_s$ that share a common factor x of length $k - 1$ where $w_p x v_s$ is not a valid word (Rogers and Pullum, 2011, see also De Luca and Restivo, 1980). If such w and v can be found for any k, then the language is not SL.

Consider a slight modification to this example language: not only are “aa” and “bb” forbidden, but also “ada”, “bdb”, “adda”, “bddb”, etc. Essentially, “d” is invisible to the constraint. Now $w = \mathrm{ad}^{k-1}\mathrm{b}$ and $v = \mathrm{bd}^{k-1}\mathrm{a}$ are both valid words, but after suffix-substitution we have “$\mathrm{ad}^{k-1}\mathrm{a}$” which is not valid. Thus this pattern is not SL. The TSL class seeks to capture exactly the constraints that would be SL if only some category of symbols could be ignored. It is defined by applying an SL grammar not to the words themselves but to their projection to a **tier alphabet** T (Heinz et al., 2011). In this case, $\mathrm{T} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ and the grammar is the same as before. See Chapter 3 for further information on the TSL class, including how to decide membership. Note that while the previous discussion involved forbidden factors, the finiteness of both the factor width and alphabet size allows an equivalent description in terms of permitted factors.

4.1.2 Our Learning Problem

Gold (1967) introduces a number of learning paradigms, but here we will focus on the case when a language is learned in the limit from distribution-free positive data. We further restrict ourselves to consider only online learning, where the induction function takes as arguments not an entire input set, but only a single input item along with a previously proposed grammar. This section formalizes these notions.

Let $L \subseteq \Sigma^*$ be a stringset and let $L_{\circledcirc}$ be L with an adjoined element $\circledcirc$ that represents the lack of any string. A text for L is a total, surjective function $t\colon \mathbb{N} \to L_{\circledcirc}$, an infinite sequence of strings drawn from L that contains each string at least once, and may at some points present no data. Note that for any non-empty stringset, there are infinitely many possible distinct texts. The addition of $\circledcirc$ is a deviation from the original work by Gold but without it no text exists for the empty stringset (Osherson et al., 1986). For a given text t, let $\overrightarrow{t_n}$ represent the sequence $\langle t_0, t_1, \ldots, t_n\rangle$, i.e. the initial segment of t of length $n+1$. We denote the class of texts for $L_{\circledcirc}$ by $\mathbb{T}$ and the class of initial segments thereof by $\overrightarrow{\mathbb{T}}$.

A grammar is some representation of a mechanism by which the membership of a string in a stringset may be decided. Let $\mathbb{G}$ be a set of possible grammars, and let $\mathscr{L}\colon \mathbb{G} \to \mathscr{P}(\Sigma^*)$ be a function that transforms a grammar into its extension, i.e. the set of all strings that it accepts. Two grammars G_1 and G_2 are equivalent (written $G_1 \equiv G_2$) iff they are extensionally equal, i.e. $\mathscr{L}(G_1) = \mathscr{L}(G_2)$. A batch learner is then a total function $\varphi\colon \overrightarrow{\mathbb{T}} \to \mathbb{G}$. In words, a batch learner is an algorithm that takes as input an initial segment of a text and outputs a guess at the correct grammar.

Given a text t and a learner φ, we say that φ converges on t iff there is some point after which its guess never changes. Formally, that means there exists some $i \in \mathbb{N}$ and some grammar G such that for all $j \geqslant i$, it holds that $\varphi(\vec{t}_j) \equiv G$. If given any text t for a stringset L, φ converges on t to a grammar G such that $\mathscr{L}(G) = L$, then we say that φ identifies L in the limit. For any class of stringsets $\mathbb{L} \subseteq \mathscr{P}(\Sigma^*)$, we say that φ identifies $\mathbb{L}$ in the limit iff it identifies L in the limit for all $L \in \mathbb{L}$.

An **online learner** differs from a batch learner only in how it assumes the data is presented. For a batch learner, the function is of type $\varphi \colon \vec{\mathbb{T}} \to \mathbb{G}$ as discussed. On the other hand, an online learner, sometimes called an **incremental learner**, is of type $\varphi \colon \mathbb{G} \times L_{\circledcirc} \to \mathbb{G}$, taking as input a previous guess and a single data point (Jain et al., 2007). Ideally the online learner can take more input over time in an efficient way while maintaining a bounded information store.

4.1.3 String Extension Learning

Heinz (2010b) described a general algorithm for learning (among others) stringsets that can be described by a set of permitted factors of length bounded above by k. For this case $\mathbb{G} = \mathscr{P}(\Sigma^k)$. In general, given a function $f \colon \Sigma^* \to \mathscr{P}(\Sigma^k)$ that maps a string to the set of factors it contains, one can define a batch learner φ_f as follows

$$\varphi_f(\vec{t}_i) \triangleq \begin{cases} \varnothing & \text{if } i = 0,\, t_i = \circledcirc \\ f(t_i) & \text{if } i = 0,\, t_i \neq \circledcirc \\ \varphi_f(\vec{t}_{i-1}) & \text{if } i \neq 0,\, t_i = \circledcirc \\ \varphi_f(\vec{t}_{i-1}) \cup f(t_i) & \text{otherwise.} \end{cases}$$

Effectively one begins with a guess of the empty grammar, and for each string provided, this guess is updated to include all factors encountered in that string. A factor is **attested** iff it appears in some string in the input. The online variant of this same function is identical except that strings are provided one at a time.

$$\varphi_f(G, w) \triangleq \begin{cases} G & \text{if } w = \circledcirc \\ G \cup f(w) & \text{otherwise.} \end{cases}$$

If the parameter k is fixed and known, this approach identifies the k-SL class in the limit. If this holds and further the parameter T is fixed and known, this approach also identifies k-TSL$^{\mathrm{T}}$ in the limit. But in general when learning TSL generalizations, we want to be able to account for the case where T is unknown.

4.2 Deciding Salience

When learning a k-TSL$^{\mathrm{T}}$ stringset, if T is not provided then a learner must discover not only the underlying SL constraints, but also the class of symbols that are salient for these constraints. The model-theoretic view of k-TSL$^{\mathrm{T}}$ given by Lambert and Rogers (2020) makes this explicit in that the ordering relation never connects to a position containing a symbol outside of T. There is no way to even talk about the other symbols. It follows then that there is no way to restrict their occurrence, meaning they are freely insertable and deletable in all strings. This property is what allows for the salience-finding algorithm of Jardine and McMullin (2017). We discuss here a simplification of that original work.

Given the set of all factors under adjacency of width up to $k + 1$ in the stringset, a symbol x is freely insertable iff for each attested factor of width less than or equal to k, inserting x at each possible point in turn results in an attested factor one symbol wider. Similarly, x is freely deletable iff for each attested factor of width $k+1$ that contains x, the removal of each instance of x in turn results in an attested factor one symbol narrower. This is a deviation from the original work in that Jardine and McMullin use only factors of widths in the range $k \pm 1$, but the formulation here avoids making a special case of shorter words. The symbols that are not both freely insertable and deletable are the salient ones.

In summary, let $\mathcal{F}_{k+1}$ represent all attested factors of width $k+1$ or less and $\mathcal{F}_k$ represent all those of width k or less, and define for each symbol $\sigma \in \Sigma$ two sets: $\sigma_\oplus$ containing all possible factors obtained by adding a single instance of σ to $\mathcal{F}_k$, and $\sigma_\ominus$ containing all possible factors obtained by deleting a single instance of σ from $\mathcal{F}_{k+1}$. Then the set of salient symbols is

$$\mathrm{T} = \{\sigma \colon \sigma_\oplus \nsubseteq \mathcal{F}_{k+1} \text{ or } \sigma_\ominus \nsubseteq \mathcal{F}_k\}.$$

Thus to decide salience we can use exactly the SL string extension learner described in section 4.1.3 and then post-process the resulting grammar by this algorithm.

In terms of time and space complexity, this portion of the algorithm is relatively efficient. There are $|\Sigma|^k$ possible factors of width k, and thus by summation there are $\frac{|\Sigma|^{k+2}-|\Sigma|}{|\Sigma|-1}$ possible factors of width up to $k+1$. Supposing we store one bit per possible factor that represents whether or not it is attested, we require $\mathcal{O}(|\Sigma|^{k+1})$ bits. For each word, its factors can be found in linear time, and each factor can be marked as attested in this set in time logarithmic in the set's size. That is, for an input of size n, the time complexity of gathering factors to determine salience is $\mathcal{O}(nk \log |\Sigma|)$.

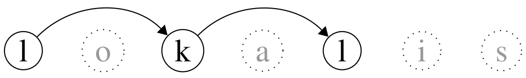


Figure 4.1: The tier-successor relation preserves linear order, but ignores certain symbols. Importantly, if a symbol is included then it is not also ignored.

4.3 The Substructures

Because Jardine and McMullin (2017) assume a batch-learning setting, they can simply learn the set of salient symbols, then erase all other symbols from the input and (in a second pass) analyze the result with any SL learner. Performing a second pass over the input requires this input to be retained. This, of course, results in unbounded space requirements and is therefore unsuitable for an online setting.

Fortunately, we can avoid this memorization of the input. A factor over relativized adjacency is made up of a sequence of symbols that appear in order, but not necessarily adjacently. These structures are, in general, referred to as **subsequences** (Heinz, 2010a; Rogers et al., 2010). Figure 4.1 shows one possible factor of width 3 ("lkl") in the string "lokalis". Crucially, when gathering subsequences, if a symbol is included in the subsequence, it can never later be excluded in that same subsequence. The factors "lki" and "lks" then would be invalid in this example and excluded from consideration because they both contain an "l" but go on to skip the second "l".

There are $\binom{n}{k}$ subsequences of width k in a word of length n, where this notation represents a binomial coefficient. It follows that the time complexity to find them is $\mathcal{O}(n^k/k!)$, and the same holds when including smaller factors as well. If for each factor we store the set of symbols that intervened, much like the paths of Jardine and Heinz (2016), then we need to account for the time it takes to find the set associated with the particular factor and to mark the intervener-set as attested.

These are additional multipliers of $k \log|\Sigma|$ and $|\Sigma|$, respectively. So in total the time complexity is $\mathcal{O}\big(n^k/(k-1)! \cdot |\Sigma| \log|\Sigma|\big)$.

Formally, if $w = \sigma_1 \dots \sigma_n$ ($\sigma_i \in \Sigma$) is a string and $X = \langle i_1, \dots, i_k \rangle$ ($k \leqslant n$) is an increasing sequence of indices, then the subsequence indicated by X is $Q = \langle \sigma_{i_1}, \dots, \sigma_{i_k} \rangle$. The intervener-set is $\mathcal{I} = \{\sigma_j \colon i_1 < j < i_k \text{ and } j \notin X\}$. The pair $\langle Q, \mathcal{I} \rangle$ is the **augmented subsequence** indicated by X. A **valid subsequence** is one where no symbol appears in both Q and $\mathcal{I}$. Henceforth, any mention of subsequences is restricted to the valid ones.

Unfortunately it appears that to store all possible augmented subsequences, we would need space significantly beyond exponential in the size of the alphabet and factor width, $\mathcal{O}(|\Sigma|^k \cdot 2^{|\Sigma|})$. But it turns out that we can exploit some structure in order to store significantly less. If a subsequence is in fact contiguous, that is it skips nothing, then no matter how adjacency is relativized that subsequence will still be an attested factor as long as it is valid. In fact a generalization holds: if a factor is attested with intervener-set $\mathcal{I}$, then it can also be assumed to be attestable for any superset of $\mathcal{I}$ for which it remains valid. So one needs only maintain the smallest observed intervener-sets (partially-ordered by subset). This means that the size of the set stored by any particular factor will never exceed $\mathcal{O}\big(\binom{|\Sigma|}{|\Sigma|/2}\big)$, which is still exponential in $|\Sigma|$, but many factors will store just a single set: $\varnothing$. Given these interactions, we conjecture that the space complexity will often be sub-exponential in the size of $|\Sigma|$. Table 4.1 shows the possible augmented subsequences for $k = 2$ in the string “cabacba” and indicates which subset of those actually need to be maintained. However, we show in section 4.4 that we can avoid this source of space complexity entirely.

Table 4.1: In gathering augmented subsequences for “cabacba”, many possibilities can be ignored. The intervener-sets are shown simply as sorted strings to avoid nested braces. Here, only the undecorated sets are maintained; those struck through were invalid from the start, while those in light gray are subsumed.

Factor	Intervener-Sets
aa	{b, ~~abc~~, bc}
ab	{∅, ~~abc~~, c}
ac	{~~ab~~, ∅}
ba	{∅, ~~abc~~}
bb	{ac}
bc	{a}
ca	{∅, ~~ab~~, ~~abc~~, b}
cb	{a, ~~abc~~, ∅}
cc	{ab}

Once the set of salient symbols T is known, we can derive a standard k-TSL$^{\mathrm{T}}$ grammar from this set of augmented subsequences. A subsequence is a permitted factor iff all of the symbols that comprise it are salient and it is attested for an intervener-set that is disjoint with T. Otherwise it is a forbidden factor.

4.4 Pointwise String Extension Learning

In sections 4.2 and 4.3, we discussed two different kinds of substructure that can be gathered when learning k-TSL: the substrings of length bounded above by $k+1$, which allow us to determine which symbols are salient, and the augmented subsequences of length bounded above by k, which allow us to select a set of permitted factors once salience has been determined. If we let the hypothesis space $\mathbb{G} = \mathscr{P}(\Sigma^{\leqslant k+1}) \times \mathscr{P}(\Sigma^{\leqslant k} \times \mathscr{P}(\Sigma))$, then we can define a learner

$$\varphi(\langle G_\ell, G_s\rangle, w) \triangleq \begin{cases} \langle G_\ell, G_s\rangle & \text{if } w = \circledcirc \\ \big\langle G_\ell \cup f(w), r\big(G_s \cup x(w)\big)\big\rangle & \text{otherwise,} \end{cases}$$

where $f\colon \Sigma^* \to \mathscr{P}(\Sigma^{\leqslant k+1})$ gathers all and only those substrings of w whose width is bounded above by $k+1$, $x\colon \Sigma^* \to \mathscr{P}(\Sigma^{\leqslant k} \times \mathscr{P}(\Sigma))$ extracts the valid augmented subsequences of w of width bounded above by k, and $r\colon \mathscr{P}(\Sigma^{\leqslant k} \times \mathscr{P}(\Sigma)) \to \mathscr{P}(\Sigma^{\leqslant k} \times \mathscr{P}(\Sigma))$ restricts the set of augmented subsequences to exclude any that are entailed by any other. This is effectively two distinct string extension learners run in parallel, pointwise on the composite grammar.

The composite grammar can immediately be used as an acceptor without further processing. We replace the cost of deciding salience by that of finding augmented subsequences.

$$\mathscr{L}(\langle G_\ell, G_s\rangle) \triangleq \bigl\{w\colon f(w) \subseteq G_\ell \text{ and } r\bigl(G_s \cup x(w)\bigr) \subseteq G_s\bigr\}.$$

In words, a string is accepted iff it has only permitted substrings and each of its valid augmented subsequences is attested or entailed by something that is attested.

Depending on the parameters and the size of the input words, this strategy might be a good one. In other situations, it might be better to actually determine which symbols are salient. Recall that a text contains every valid word at least once, and that non-salient symbols are freely deletable in all strings. Free deletability of non-salient symbols implies that any subsequence that includes only salient symbols will, in some word of the text, have only its salient interveners as interveners. Those subsequences that do not violate constraints on the tier then must appear with an empty intervener set. In other words, such subsequences will appear as factors in terms of adjacency and will be accounted for by that component of the composite grammar.

Thus upon convergence the left component of this composite grammar, G_ℓ, is sufficient on its own to decide salience and to extract the grammar itself. G_s is unnecessary. Let $s\colon \mathbb{G} \to \mathscr{P}(\Sigma)$ be the function that decides salience in the manner described in section 4.2, and let $\pi_{\mathrm{T}}(w)$ represent the projection to T of w as described by erasing symbols that are not in T. Then we might have the following as an equivalent alternative definition

$$\mathscr{L}(G) \triangleq \bigl\{w\colon f\bigl(\pi_{s(G)}(w)\bigr) \subseteq G\bigr\}.$$

As an aside, the deletion-closure of the strictly piecewise class of stringsets (Rogers et al., 2010, see also Haines, 1969) enables this same sort of learning of long-distance patterns using only adjacent substrings.

Revisiting the time and space complexities mentioned previously, this optimized version of the grammar can be learned in $\mathcal{O}(nk\log|\Sigma|)$ time and $\mathcal{O}(|\Sigma|^k)$ space, exactly those values that were assumed for only the salience-decision component of learning. The time complexity is deferred to later, in interpreting the grammar as k-TSL rather than as $(k+1)$-SL.

4.5 A Worked Example of the Final Simplified Approach

Consider a blocked-assimilation constraint such as the liquid dissimilation of Latin (Cser, 2010), where identical liquids may not occur in sequence unless a non-coronal intervenes. This is equivalent to the example language described in section 4.1.1. We will assume for now that no other constraint interacts with this. Assuming an alphabet consisting of two distinct liquids (“l” and “r”), a non-coronal consonant (“k”), and a vowel (“a”), we discuss a worked example that learns this constraint.

Table 4.2: Some words of varying degrees of phonological plausibility. Each set is in the order produced by a sliding 3-window. Factors already accounted for are in light gray.

akkalkak	{akk, kka, kal, alk, lka, kak}
klark	{kla, lar, ark}
kralk	{kra, ral, alk}
karlakalra	{kar, arl, rla, lak, aka, kal, alr, lra}
akrala	{akr, kra, ral, ala}
aklara	{akl, kla, lar, ara}
rakklarkka	{rak, akk, kkl, kla, ark, rkk, kka}
arkralkla	{ark, rkr, kra, ral, alk, lkl, kla}
laarlraalr	{laa, aar, arl, rlr, lra, raa, aal, alr}
kaaakkrka	{kaa, aaa, aak, akk, kkr, krk, rka}
klkkklrk	{klk, lkk, kkk, kkl, klr, lrk}
krlkrkl	{krl, rlk, lkr, krk, rkl}
alrla	{alr, lrl, rla}

As this can be described by the 2-TSL$^{\{\mathrm{l,k,r}\}}$ constraint whose forbidden factors are $\{\mathrm{ll}, \mathrm{rr}\}$, the text must contain all substrings of width 3 or less whose projection to $\{\mathrm{l}, \mathrm{k}, \mathrm{r}\}$ do not contain these prohibited bigrams. The words shown in Table 4.2 would constitute a representative sample for this constraint, though one may notice that some of the 3-factors that need to appear are phonologically implausible.

If we assume that domain boundaries are explicit, then we would also need to encounter any permitted factors that include these boundary symbols. For the sake of brevity such an account has been omitted, but one could easily construct similarly implausible words to account for this change.

4.6 Non-Strict Locality

These methods can be extended beyond just TSL. Chapter 3 demonstrates that the locally testable (McNaughton and Papert, 1971) and locally threshold testable (Beauquier and Pin, 1989) stringsets admit T-relativized variants with the same properties as TSL$^{\mathrm{T}}$:

- A string appears iff its projection to T appears, and
- All symbols that are not in T are freely insertable and deletable.

Locally threshold testable stringsets are characterized by not just the factors in each word but the saturating multisets of factors in the words. A **saturating multiset** is a variant of the multiset in which the counts associated with elements are capped to some maximum value, t. Multisets that saturate at a count of $t = 1$ are simply sets, and these are the characterization of locally testable.

Due to the two properties of relativization, if we collect the saturating multisets of substrings of width bounded above by k along with the individual factors of width $k + 1$, we can still use the algorithm described in section 4.2 to determine salience and again treat the non-relativized grammar in a relativized way. The issue is not finding a learning algorithm; instead it is the space complexity. Figure 4.2 shows the amount of space required for factors, $\mathcal{O}(|\Sigma|^k)$, compared to that of saturating multisets, $\mathcal{O}\big((t+1)^{|\Sigma|^k}\big)$. Of course, Σ and k are fixed parameters for any given run of the algorithm, and thus constant, but this complexity should not be ignored.

4.7 Conclusions

We proposed an online learning algorithm for the tier-based strictly k-local class of stringsets that operates in linear time, $\mathcal{O}(nk \log |\Sigma|)$, and constant space, $\mathcal{O}(|\Sigma|^{k+1})$, in terms of the size of the input. This space complexity is exponential in the factor width. We demonstrated that the grammar representation given by a strictly k-local learner can also be interpreted as a strictly k-piecewise or tier-based strictly $(k-1)$-local grammar. The difference comes later, in the interpretation of

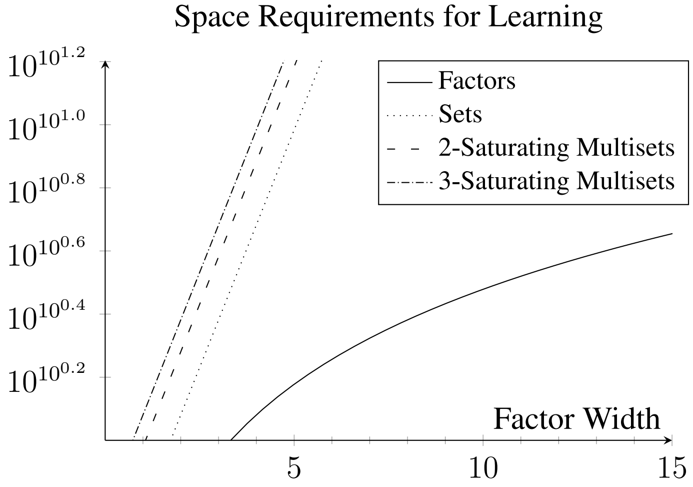


Figure 4.2: While gathering factors requires space exponential in terms of factor width, the requirements are doubly exponential for any of the larger structures we might employ. Here the space requirements are shown for just a binary alphabet.

the grammar. The algorithms presented here can be incorporated into any sufficiently general implementation of string extension learners (Heinz, 2010b).

Efficient learning of interacting constraints remains an open question. Generally the set of symbols salient to a pattern as a whole will be some superset of those sets for its constraints. If a stringset L consists of a TSL component with an additional constraint imposed that restricts the set of substrings that may occur, then L will not in general be learnable by the known TSL-learning algorithms including those presented here. If multiple TSL constraints over different tier alphabets interact, the learned stringset will consider the set of salient symbols to be the union of all such alphabets, but other sorts of interactions have yet to be explored. Notably, SL is equivalent to TSL^{Σ} by definition, and any SP constraint imposes a tier of salience including the symbols that it mentions, so many cases of interaction will result in a TSL^{Σ} (that is, SL) approximation of the target stringset.

This lack of robustness in the face of constraint interaction poses a challenge for the learnability of

TSL constraints within a more complex structure in a natural setting. If the solution to this problem is learning a new grammar for each possible tier alphabet, rather than trying to determine which such alphabet to consider, then we must bear in mind the additional space requirements, exponential in the size of the alphabet. Such an approach yields the multiple-tier-based strictly local languages of De Santo and Graf (2019).

Chapter 5: Monoid Varieties and a Subregular Spiral

One of the primary questions in formal language theory in general and in linguistics specifically is on what mechanisms are necessary and sufficient to classify words in a language (Beauquier and Pin, 1991; Rogers et al., 2012), or to learn a language (Valiant, 1984; Heinz et al., 2012; Lai, 2015).

The local hierarchy of classes discussed by McNaughton and Papert (1971) and extended by Beauquier and Pin (1989) combines with the piecewise testable class studied by Simon (1975) and its strict subclass from Rogers et al. (2010, see also Haines, 1969) to form the piecewise-local subregular hierarchy. In order to more naturally account for phonological descriptions, Heinz et al. (2011) introduced the tier-based strictly local languages, then defined operationally but since characterized language- and model-theoretically by Lambert and Rogers (2020). Learning algorithms for each of these classes are well-established (Heinz, 2010b; Heinz et al., 2012; Lambert, 2021a).

These are not the only classes of formal languages, however. Extensive study has gone into classes of languages characterized by definability in various logics (Elgot, 1961; Thérien and Wilke, 1998; Grädel and Otto, 1999; Weis and Immerman, 2009; Krebs et al., 2020). Connections between first-order logic and star-free languages (Thomas, 1982), monadic second-order and regular (Büchi, 1960) have been established, and fragments of first-order logic describe piecewise-local classes (Rogers and Lambert, 2019a) and others (Thérien and Wilke, 1998).

Language-theoretic characterizations are nice because one can quickly, with just a few example words, show that a given language cannot belong to a given class. Logical, model-theoretic characterizations are nice because one can show simply by providing a formula that a language must be in a given class. However, with either it may be more difficult to come the opposite conclusion. With the language-theoretic approach it may be difficult to partition the entire possibly infinite language in order to assert the truth of the necessary properties, and with the logical approach one often must resort to Ehrenfeucht-Fraïssé games to prove inexpressibility (Libkin, 2004). An algebraic connection allows for both affirmation and denial by testing only a few properties.

This paper uses the algebraic approach to unite the factor-based piecewise-local hierarchy often invoked in linguist study with the purely logical language classes. A general mechanism for deciding whether a language class admits a tier-based variant is discussed, and classes neglected or forgotten are incorporated to complete a grid. The structure of the paper is as follows: the next section includes background material on algebraic structures, and a brief discussion on why it is useful to begin analysis with a variety of monoids rather than just some arbitrary collection of properties. Then section 5.2 discusses a general notion of when a language class admits a tier-based relativization. Section 5.3 discusses some specific classes, relating them to one another as they are introduced. Finally, a summary of the classes and their relationships is provided.

5.1 Background

A semigroup is a set S coupled with an associative binary operation $\cdot$ which is often denoted implicitly by adjacency. The free semigroup over a set G of generators is the infinite set of all nonempty sequences over G with concatenation as the operation. A monoid is a semigroup with an identity element, 1, and the free monoid over G is the free semigroup augmented with the empty sequence, often denoted ε.

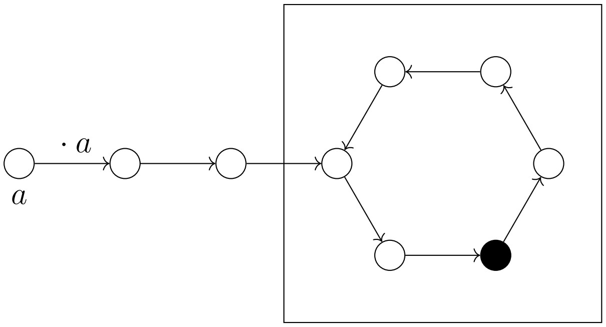


Figure 5.1: Every element a of a finite semigroup S eventually generates a group.

In terms of notation, Σ represents some finite alphabet, while Σ^* and Σ^+ represent the free monoid and free semigroup over Σ, respectively. A formal language is some subset of Σ^*. For a given formal language L, the Myhill equivalence class for that language, denoted $\sim_L$, is defined as $u \sim_L v$ iff for all $x, y \in \Sigma^*$ it holds that $xuy \in L$ when and only when $xvy \in L$. It is a theorem that L is regular iff there are finitely many equivalence classes under this relation (Rabin and Scott, 1959). A coarser partition defines the minimal finite-state acceptor for L (Nerode, 1958), and in fact if such an acceptor has q states then the Myhill relation may induce up to q^q equivalence classes (Holzer and König, 2004). But this relation is nice in that it forms a **congruence**, an equivalence relation where the monoid operation yields equivalent elements when applied to equivalent elements: if $u \sim_L u'$ and $v \sim_L v'$ then $uv \sim_L u'v'$. The equivalence classes can be concatenated consistently, and so these equivalence classes form a quotient monoid. This is known as the **syntactic monoid** of L and is denoted $M(L) = \Sigma^*/\sim_L$. Its **syntactic semigroup** is $S(L) = \Sigma^+/\sim_L$. Notably, a language L and its complement $\complement L$ share the same structure.

Because all regular languages have a finite syntactic monoid, we concern ourselves here only with these finite monoids. One interesting property of a finite monoid is that for all elements a, there exists some m and some $n > m$ such that $a^m = a^n$. Figure 5.1 shows a subsemigroup generated by a where $m = 4$ and $n = 10$. The boxed elements form a group whose identity is a^6, as marked. The cycle length is $n - m$, and the identity of the cyclic group formed is $a^{m(n-m)}$. Denote $a^{m(n-m)}$ by a^ω; it is the unique power of a that is idempotent. Again, such a power always exists in a finite semigroup. A language is star-free iff all of these generated cyclic groups are **trivial**, containing only a single element (Pin, 1997). In other words, for a star-free language there is some m such that $a^m = a^{m+1}$ for all a (Schützenberger, 1965).

A variety $\mathbf{V}$ of finite monoids is a class of finite monoids that is closed under taking submonoids, quotients, and finite Cartesian products. Properly, this is a pseudovariety (Eilenberg and Schützenberger, 1976), but the restriction to finite structures is justified for the study of subregular languages. A variety of finite semigroups is defined similarly. Several classes of formal languages are characterized by the syntactic monoids or semigroups of their languages belonging to some variety. One specific example is the variety of **bands**, which consists of those semigroups whose operation is **idempotent**, where $x \cdot x = x$ for all x. Another is the variety of **semilattices**, $\mathbf{J_I}$, which consists of commutative bands, where additionally $x \cdot y = y \cdot x$ for all x and y. The aperiodic (star-free) structures form the variety $\mathbf{Ap}$.

Monoid pseudovarieties are quite useful in characterizing language classes. If A is a formal language, then $M(A)$ is the smallest monoid that can recognize the languages. That is, there are certain elements x such that a word w is accepted iff it is in the equivalence class denoted by one of these x. The syntactic monoid can be viewed as a possibly nonminimal string acceptor. If A and B are two formal languages, then the familiar product construction (see Rabin and Scott, 1959; Hopcroft and Ullman, 1979) can be applied to their syntactic monoids, resulting in a new monoid, the Cartesian product of $M(A)$ and $M(B)$. This recognizes $A \cap B$ or $A \cup B$ depending on how parity is assigned. Because monoid pseudovarieties are closed under products, the new monoid satisfies the same properties. However, this may not be the smallest such monoid. It can however be minimized by a quotient: $(M(A) \times M(B))/\sim_{A \otimes B}$, where $\otimes$ is either union or intersection. Because pseudovarieties are closed under quotient, the same properties remain satisfied even over the smaller structure. Recalling that a language and its complement share a monoid, we see that if $\mathbb{C}$ is a class of languages defined by a pseudovariety $\mathbf{V}$ of monoids, then $\mathbb{C}$ is closed under the Boolean operations.

5.1.1 Green's Relations

Green (1951) defined several relations that have become incredibly useful. For a semigroup S, denote by S^1 the monoid formed by adding a new element 1 to S in the case that it does not already have an identity element, or S itself if it does. The principle left ideal generated by a is $S^1a = \{sa\colon s \in S^1\}$. The principle right ideal and principle two-sided ideal are aS^1 and S^1aS^1, respectively. The relations are as follows: $a\mathscr{L}b$ iff $S^1a = S^1b$, $a\mathscr{R}b$ iff $aS^1 = bS^1$, $a\mathscr{J}b$ iff $S^1aS^1 = S^1bS^1$. In other words, elements are related if they generate the same left, right, or two-sided ideals, respectively. These also generate preorders, where $a \leqslant_{\mathscr{L}} b$ iff $S^1a \subseteq S^1b$ and so on. Then $a\mathscr{H}b$ iff $a\mathscr{L}b$ and $a\mathscr{R}b$, while $a\mathscr{D}b$ iff for some $c \in S$ it holds that $a\mathscr{L}c$ and $c\mathscr{R}b$. If M is a monoid and x an element, define $M_x = \{g\colon x \leqslant_{\mathscr{J}} g\}$, the set of all g such that for some a and some b, $agb = x$.

If $\mathbf{V}$ is a variety, then a new variety $\mathbf{M_eV}$ can be defined to contain all and only those monoids for which $eM_ee \in \mathbf{V}$ for all idempotents e. For example, the variety $\mathbf{1}$ of trivial monoids, which characterizes $\Sigma^* \cup \varnothing$, can be extended to $\mathbf{M_e 1}$. Further one can define the variety $\mathbf{LV}$, where a semigroup S is in $\mathbf{LV}$ iff for all its idempotents e it holds that eSe, the **local subsemigroup** associated with e, is in $\mathbf{V}$. It is easy to show that both $\mathbf{LV}$ and $\mathbf{M_e V}$ truly are varieties.

A monoid is said to be **trivial under** a relation iff its equivalence classes under that relation are all singleton. The following theorem motivates naming the class of semilattices $\mathbf{J_I}$.

Theorem 7. *A monoid M is a semilattice iff it is $\mathscr{J}$-trivial and idempotent.*

Proof. Suppose M is a semilattice and let e be an element of M. Now suppose a and b are elements of M_e. There must exist u, v, x, and y such that $uav = xby = e$. Then $aeb = aeeb = auavxbyb = ua^2vxb^2y = uavxby = ee = e$. Thus all semilattices are $\mathscr{J}$-trivial and idempotent.

Now to show the reverse. Suppose instead that M is $\mathscr{J}$-trivial and idempotent, and let e and f be elements of M. Then $ef = eff$ and $f \in M_e$. Because M is $\mathscr{J}$-trivial then, $fef = e$. It follows that $ef = feff = ffef = fe$. Therefore all monoids both $\mathscr{J}$-trivial and idempotent are semilattices. ■

5.2 Tier-Based Classes of Languages

The tier-based strictly local (TSL) class of formal languages was defined operationally by Heinz et al. (2011). Essentially, a language L is TSL iff there is some **tier** T of salient symbols such that the projection of L to T, removing symbols not in T, is strictly local (SL). It was characterized language-theoretically by Lambert and Rogers (2020). Specifically it is noted that the nonsalient symbols can be freely inserted and deleted. In other words, for all $u, v \in \Sigma^*$ and for all $\tau \notin \mathrm{T}$ $u\tau v$=uv. This means that $\tau \sim_L \varepsilon$, where ε refers to the empty string.

Lemma 2. *If M is the syntactic monoid of a star-free language and $a, b \in M$ such that $ab = 1$, then $a = b = 1$.*

Proof. Suppose a and b are elements of M such that $ab = 1$ and that M is the syntactic monoid of a star-free language. Then $1 = ab = (a(ab))b$. By continuing this process, we see that $a^n b^n = 1$ for all n. Because M is star-free, there exists some m such that $a^m = a^{m+1}$ and therefore $1 = a^m b^m = aa^m b^m = a$. Similarly, $1 = b$. Therefore $a = b = 1$. ■

This means that for any star-free language L, if $w \sim_L \varepsilon$ then for all $\sigma \in \Sigma$ it holds that if $w = a\sigma b$ for some a and some b then $\sigma \sim_L \varepsilon$. The set of symbols salient to the language is $\mathrm{T} = \{\sigma \colon \sigma \in \Sigma, \sigma \not\sim_L \varepsilon\}$. Define the **projected subsemigroup** of L as $X(L) = \mathrm{T}^+ / \sim_L$.

Theorem 8. *If L is a star-free language whose syntactic monoid is nontrivial, then $X(L)$ is isomorphic to $M(L) - \{1\}$.*

Proof. Suppose L is star-free and $s \in M(L)$. Then there is some $w \in \Sigma^*$ such that the s is the equivalence class containing w.

Suppose $s = 1$. Then $w \sim_L \varepsilon$ and it follows from Lemma 2 that $w \in \complement\mathrm{T}^*$, and therefore no equivalence class in $\mathrm{T}^+ / \sim_L$ contains w. Therefore $1 \notin X(L)$.

On the other hand, suppose $s \neq 1$. Then $w = \sigma_1 \cdots \sigma_n$ for $\sigma_i \in \Sigma$. Define π_{T} as follows:

$$\pi_{\mathrm{T}}(w) = \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \sigma\pi_{\mathrm{T}}(v) & \text{if } w = \sigma v \text{ for } \sigma \in \Sigma \text{ and } \sigma \not\sim_L \varepsilon \\ \pi_{\mathrm{T}}(v) & \text{if } w = \sigma v \text{ for } \sigma \in \Sigma \text{ and } \sigma \sim_L \varepsilon. \end{cases}$$

Then $\pi_{\mathrm{T}}(w) \sim_L w$, as the only elements removed were identities, but because it is not itself the identity, $\pi_{\mathrm{T}}(w) \in \mathrm{T}^+$. Then $s \in X(L)$.

It follows then that $X(L)$ is identical to $M(L)$ minus any identity element. ■

If $\mathbb{C}$ is a nontrivial class of languages characterized by the syntactic semigroups of its languages, then there exists a class $\mathrm{T}\mathbb{C}$, the tier-based $\mathbb{C}$ class, characterized by its projected subsemigroups. However, when we are dealing in monoids and the empty string is included, projection cannot remove 1 from the structure. So if $\mathbb{C}$ were instead characterized by its syntactic monoids, then $\mathrm{T}\mathbb{C}$ would be equivalent to $\mathbb{C}$ itself. It follows then that if another class $\mathbb{C}'$ properly contains $\mathbb{C}$ but is characterized by its monoids, then $\mathbb{C}'$ also contains $\mathrm{T}\mathbb{C}$. The following sections explore some other classes, eventually resulting in a hierarchy of inclusion.

5.3 Specific Classes of Formal Languages

Stepping back for a moment, formal logic has been used to define several classes of languages. Rogers and Lambert (2019a) discuss a quantifier-free logic where a **factor** is a sequence over Σ^* and is satisfied by all words that contain that factor. What containment is varies depending on how it is interpreted: a **local factor** f is satisfied by all words w such that $w = xfy$ for some x and some

y, while a **piecewise factor** $\sigma_1 \cdots \sigma_n$ is satisfied by all words w such that $w = x_1\sigma_1 \cdots x_n\sigma_n x_{n+1}$ for $x_i \in \Sigma^*$. A local factor can be written $\sigma_1 \lhd \cdots \lhd \sigma_n$ while a piecewise factor can be written $\sigma_1 < \cdots < \sigma_n$. More generally, a factor can be written as a pair $\langle \sigma_1 \cdots \sigma_n, R_1 \cdots R_{n-1}\rangle$ where the σ_i are in Σ and the R_i are ordering relations such as $\lhd$ or $<$. Such a factor can be expanded into a first-order sentence over two variables by two mutually recursive functions, defined here by v ranging over the values x and y and $\overline{v}$ representing the other one:

$$\varphi_v(w,\rho) = \begin{cases} \top & \text{if } w = \varepsilon \\ \sigma(v) & \text{if } w = \sigma \text{ and } \rho \text{ is empty} \\ \sigma(v) \wedge (\exists\overline{v})[v\ r\ \overline{v} \wedge \varphi_{\overline{v}}(u,s)] & \text{if } w = \sigma u \text{ and } \rho = rs. \end{cases}$$

Concretely, the factor $\langle abc, \lhd{<}\rangle$ expands to

$$\begin{aligned} (\exists x)[\varphi_x(abc, \lhd{<})] &= (\exists x)[a(x) \wedge (\exists y)[x \lhd y \wedge \varphi_y(bc, <)]] \\ &= (\exists x)[a(x) \wedge (\exists y)[x \lhd y \wedge b(y) \wedge (\exists x)[y < x \wedge \varphi_x(c, \varepsilon)]]] \\ &= (\exists x)[a(x) \wedge (\exists y)[x \lhd y \wedge b(y) \wedge (\exists x)[y < x \wedge c(x)]]] \end{aligned}$$

Indeed, the meanings of these factors are definable in first-order logic restricted to two variables, so long as the necessary ordering relations are supplied. This is referred to as FO$^2[R_1, \ldots, R_n]$ where the R_i are ordering relations. For example, local factors, which only adjacency, are definable in FO$^2[\lhd]$, while piecewise factors are definable in FO$^2[<]$. Languages definable by a Boolean combination of local factors are **locally testable** (LT). while those definable by a Boolean combination of piecewise factors are **piecewise testable** (PT).

5.3.1 Piecewise Testable

The piecewise testable languages were explored by Simon (1975). In short, they are all and only those languages whose syntactic monoids are $\mathscr{J}$-trivial. Denote by **J** this class. That is, for any PT language L, $M(L)/\mathscr{J}$ is isomorphic to $M(L)$. Later it will be relevant that $M \in \mathbf{J}$ iff for all idempotents e, $M_e e M_e = e$ (Brzozowski and Fich, 1984). Because this is a characterization on monoids, there is no distinct tier-based piecewise testable class.

Recall that a language and its complement share a structure. The languages piecewise testable in the strict sense (strictly piecewise) discussed by Rogers et al. (2010) then cannot be characterized by a their semigroups alone. By using the ordered semigroups discussed by Pin (1997), the state parity can be maintained and the the wholly nonzero property used by Fu et al. (2011) can be enforced. Such structures will not be discussed further here, however.

5.3.2 Semilattice Languages

The class CB (for commutative band) of languages whose syntactic monoids are semilattices ($\mathbf{J_I}$) is a subclass of the PT languages. A language L is in CB iff for all u, a, b, and v in Σ^* it holds that uav is in the language iff $uaav$ is as well, and $uabv$ is in the language iff $ubav$ is as well. That is, if $w \in \Sigma^*$ and v contains the same set of letters as w, then $w \in L$ iff $v \in L$. In other words, CB is equivalent to 1-PT.

There are, of course, PT languages that are not CB; consider the language over $\Sigma = \{a, b\}$ in which no word contains two instances of b. Then aba is in the language, but $abba$ is not in the language, and therefore $b \not\sim_L bb$. The equivalence class of b then is not idempotent and the languages is not in CB.

5.3.3 Locally Testable

The locally testable languages were first discussed by McNaughton and Papert (1971) and later characterized algebraically independently by Brzozowski and Simon (1973). and by McNaughton (1974). A language L is LT iff for every idempotent e of its syntactic semigroup $S(L)$, it holds that $eS(L)e$ is a semilattice. In other words, a language is LT iff its syntactic semigroup is in $\mathbf{LJ_I}$ Such local subsemigroups are always monoids, as e itself acts as an identity. Because the class of semilattices is a variety, and thus closed under the taking of submonoids, CB is a subclass of LT as well. It is equal to 1-LT. The language over $\Sigma = \{a, b\}$ that forbids bb substrings separates full LT from CB, by the same logic as in the case of PT separation. Further, because LT is characterized by its semigroups rather than by its monoids, there is a distinct tier-based locally testable class, and a language forbidding ab substrings on the $\{a, b\}$ tier separates them when $\Sigma = \{a, b, c\}$.

The piecewise testable class is incomparable with the (tier-based) locally testable class, as the language forbidding the piecewise factor $a < b < c$ is in the former but not the latter while the language forbidding the local factor $a \lhd b \lhd c$ is in the latter but not the former.

Like the strictly piecewise languages, the strictly local languages are not closed under complementation and thus cannot be characterized by their monoids alone. De Luca and Restivo (1980) provide a characterization nonetheless, again essentially by preserving state parity.

5.3.4 Generalized Definite

A subclass of locally testable languages are those where the syntactic semigroup is locally trivial, $\mathbf{L1}$. That is, for all idempotents e, $eSe = e$. These are the generalized definite languages, GD

(Brzozowski and Fich, 1984). This is a semigroup property, so a distinct tier-based extension exists. Over $\Sigma = \{a, b\}$, the language in which no word contains two instances of a is TGD but not GD or even LT, affirming the distinction. The language where all words contain some ab substring is LT but not GD, so the containment is strict. Note that while the definite and reverse-definite languages are subclasses of the strictly local class, the generalized definite languages are not.

The (tier-based) generalized definite class is incomparable with the piecewise testable languages. The language wherein all words begin with ab is in the former but not the latter, while the language forbidding the presence of an a that is eventually followed by a b is in the latter but not the former.

5.3.5 Trivial Languages

An even smaller class is that of trivial languages, those whose syntactic monoid contains only a single element. The variety of trivial monoids is called $\mathbf{1}$, and the associated class of languages is denoted 1. For each word w, it holds that $w \sim_L \varepsilon$. Thus, the language must be empty or it must contain all possible words in Σ^*. This is a monoid-based characterization, so no distinct tier-based variant exists.

An interesting effect surfaces if we look at semigroup-trivial languages. As this class is not closed under products, it is not a pseudovariety of monoids and thus not a natural foundation for exploration. But it does demonstrate a strange property of tier relativization. L is semigroup-trivial iff L is Σ^*, Σ^+, or the complement of one of these. For a given alphabet, there are $|\Sigma| - 1$ distinct stacked tier-based extensions of this class. For instance, the language defined by the following regular expression over $\Sigma = \{a, b, c, d\}$ is $\mathrm{TTT}1_S$ but not $\mathrm{TT}1_S$:

$$(d^*c[cd]^*b + d^*b)[bcd]^* + dd^*$$

and such a language can be generated for any alphabet. Define an ordering of the alphabet and let Σ_i denote its first i symbols, then define the following:

$$L_1 = \Sigma_1^+$$
$$L_{i+1} = \Sigma_{i+1}^+ \wedge \neg [\![L_i]\!]_{\Sigma_i},$$

where $[\![C]\!]_\mathrm{T}$ means that constraint C applies on tier T. Then $L_{|\Sigma|}$ is $\mathrm{T}^{|\Sigma|-1}1_S$ but not $\mathrm{T}^{|\Sigma|-2}1_S$.

5.3.6 Locally Threshold Testable

There is nowhere lower in the hierarchy to descend, so let us now continue upward. Languages defined based on the occurrence of local factors whose multiplicity is counted up to some threshold t are the **locally threshold testable** languages explored by Beauquier and Pin (1989). Note that thresholded multiplicity of factors can be reduced to precedence. For multiplicity of 1, we have $\psi_v(\langle ab, 1\rangle) = \varphi_v(ab, \lhd)$ and for larger multiplicities:

$$\begin{aligned}(\exists v)[\psi_v(\langle ab, n\rangle)] = (\exists v)[a(v) \wedge (\exists \overline{v})[v \lhd \overline{v} \wedge \varphi_{\overline{v}}(b, \lhd)]\\ \wedge (\exists \overline{v})[v < \overline{v} \wedge \psi_{\overline{v}}(\langle ab, n-1\rangle)]].\end{aligned}$$

In other words, LTT is $\mathrm{FO}^2[<, \lhd]$-definable.

These languages are characterized by aperiodic (star-free) semigroups where for all idempotents e and f and for all elements a, b, and u, it holds that $eafuebf = ebfueaf$. Because this is a characterization on semigroups, there is a distinct tier-based locally threshold testable class, and a language forbidding occurrences of an ab substring on the $\{a, b\}$ tier separates them when $\Sigma = \{a, b, c\}$. The TLTT and PT classes are incomparable: if $\Sigma = \{a, b, c\}$, then the language forbidding three or more occurrences of an ab substring is in the former but not the latter, while the language forbidding the abc piecewise factor is in the latter but not the former.

5.3.7 FO$^2[<]$ and $\mathscr{G}$-Triviality

The variety $\mathbb{D}\mathbf{A}$ of semigroups is defined such that $(xyz)^\omega y(xyz)^\omega = (xyz)^\omega$ (Thérien and Wilke, 1998). The variety of $\mathscr{G}$-trivial monoids are those where for some idempotent e, the submonoid $eM_e e = e$ (Fich and Brzozowski, 1979). In other words, $\mathscr{G}$-trivial monoids make up the variety $\mathbf{M_e 1}$.

Theorem 9. $\mathbb{D}\mathbf{A}$ *and $\mathscr{G}$-trivial are equivalent for finite structures.*

Proof. Suppose S is a semigroup in $\mathbb{D}\mathbf{A}$ and let $M = S^1$. Let e be an idempotent of S. Then $M_e = \{g \colon agb = e \text{ for some } a, b\}$. Now by assumption for all $g \in M_e$ it holds that $ege = (agb)^\omega g(agb)^\omega = (agb)^\omega = e^\omega = e$, so $eM_e e$ is the singleton monoid containing only e. Thus all semigroups in $\mathbb{D}\mathbf{A}$ are $\mathscr{G}$-trivial.

Instead suppose M is a $\mathscr{G}$-trivial monoid. Then for all idempotents e, $eM_e e = e$. By definition, $\chi = (xyz)^\omega$ is an idempotent for all x, y, and z. Then y is in M_χ, as $(x) \cdot y \cdot (z(xyz)^{\omega-1}) = (xyz)^\omega = \chi$.

But because χ is idempotent, it follows from $\mathscr{G}$-triviality that $\chi y \chi = \chi$. Because there was no restriction on x, y, or z, it follows then that any $\mathscr{G}$-trivial M is in $\mathbb{D}\mathbf{A}$.

Thus the two are equivalent. ■

Thérien and Wilke (1998) show that a language is first-order definable with two variables and general precedence iff its syntactic semigroup is in $\mathbb{D}\mathbf{A}$. This holds iff its syntactic monoid is $\mathscr{G}$-trivial. Because $\textsc{fo}^2[<]$ is characterized by a monoid property, there is no distinct tier-based extension of the class.

Because eM_ee is always a submonoid of eSe, it is necessarily the case that if $M \in \mathbf{LV}$ for some variety $\mathbf{V}$, then $M \in \mathbf{M_eV}$ as well. However, we can show that the tier-based extension of $\mathbf{LV}$ is also a subclass of $\mathbf{M_eV}$.

Theorem 10. *If a variety* $\mathbf{V}$ *is a subclass of* $\mathbf{Ap}$*, the class of aperiodic semigroups, and if* $X(L) \in \mathbf{M_eV}$ *then* $M(L) \in \mathbf{M_eV}$ *as well.*

Thus if $X(L) \in \mathbf{LV}$ it must be in $\mathbf{M_eV}$ by containment and therefore the tier-based extension of the former variety is a subclass of the latter one. We must see then that TGD is a subclass of the languages definable in $\textsc{fo}^2[<]$. This is interesting, as the generalized definite languages are those in which prefix and suffix substrings, naturally described in terms of adjacency, decide whether a word may be included, but the successor relation cannot be defined in terms of general precedence when restricted to only two variables. To discuss a prefix (or suffix) of length k, one can build it as a piecewise factor using general precedence and then assert additionally that there are not k positions before its final component (or after the initial one).

Using the logical methods described earlier however, we note that all piecewise testable languages are definable in this logical language, implicitly stating that $\mathbf{J}$, the variety characterizing PT, is a subclass of $\mathbf{M_e 1} = \mathbb{D}\mathbf{A}$. This inclusion is also noted by Brzozowski and Fich (1984). This class is incomparable with LT, however, as the language forbidding occurrences of an ab substring demonstrates that the latter does not contain this class, which by inclusion of PT is not contained by LT.

5.3.8 Generalized Locally Testable

Recall that the locally testable languages are those whose syntactic monoids are in $\mathbf{LJ_I}$, requiring eSe to be a semilattice. By replacing S with M_e, we get the variety $\mathbf{M_e J_I}$ which requires that $eM_e e$ be a semilattice, and we obtain the generalized locally testable class (GLT) (Brzozowski and Fich, 1984). This is a monoid characterization, so there is no distinct tier-based GLT. It is trivially true that LT is a subclass of GLT, as S is a superset of M_e. Because languages and their tier-based extensions share a monoid it must hold that TLT is a subclass of GLT as well. Further, the class of $\mathscr{J}$-trivial monoids is also a subclass of GLT as noted by Brzozowski and Fich (1984), so the latter also contains PT. In fact FO$^2[<]$ is a subclass of GLT, because $eM_e e$ is certainly a semilattice if it contains only the single element e. However, GLT is incomparable with LTT: if the alphabet is $\{a, b, c\}$ then the language forbidding a bb substring on the $\{b, c\}$ tier is in the former but not the latter, while the language requiring at least three occurrences of an ab substring is in the latter but not the former.

5.3.9 $\text{FO}^2[<,\lhd]$ and Local $\mathscr{G}$-Triviality

Krebs et al. (2020, see also Thérien and Wilke, 1998) show that the languages first-order definable with two variables and both successor and general precedence, $\text{FO}^2[<,\lhd]$, are all and only those that are in $\mathbf{L}\mathbb{D}\mathbf{A}$. In other words, they are locally $\mathscr{G}$-trivial. It is important that this is based on the semigroup rather than the monoid, because if the identity were included then $1M1 = M$ would necessarily satisfy the property and the class would be equivalent to $\text{FO}^2[<]$. Therefore there should also be a tier-based extension of this class. The same languages that witness the incomparability of GLT and LTT serve also to show the incomparability of GLT and $\text{FO}^2[<,\lhd]$.

There is a tier-based extension of $\text{FO}^2[<,\lhd]$, but it seems that this extension is a proper subclass of $\text{FO}^2[<,\lhd^{\mathrm{T}}]$. This extension restricts the general precedence relation to those symbols on the tier, while the latter class allows unrestricted use of general precedence.

5.3.10 $\text{FO}^2[<,\mathrm{bet}]$: Adding Betweenness

Krebs et al. (2020) discuss also the class of languages definable in first-order logic restricted to two variables with general precedence as well as a betweenness relation for each symbol $\sigma \in \Sigma$, $\sigma(x,y)$, which is true iff there is no z such that $x < z < y$ and $\sigma(z)$. Much like the successor relation, this is not definable from general precedence with two-variable logic, and must be given as an axiom. With betweenness of this sort, the successor relation is definable, so $\text{FO}^2[<,\lhd]$ is a subclass of $\text{FO}^2[<,\mathrm{bet}]$: $x \lhd y$ iff $\neg\Sigma(x,y)$. The tier-successor relation can also be defined: $x \lhd^{\mathrm{T}} y$ iff $\mathrm{T}(x) \wedge \mathrm{T}(y) \wedge \neg\mathrm{T}(x,y)$. Recall that multiplicity can be reduced to precedence, so TLTT is also a subclass of $\text{FO}^2[<,\mathrm{bet}]$. In order to show that GLT is another subclass, we proceed through the algebra.

A language L is $\textsc{fo}^2[<, \mathrm{bet}]$ iff its syntactic monoid $M \in \mathbf{M_e}\mathbb{D}\mathbf{A}$. In other words, $M \in \mathbf{M_e}(\mathbf{M_e 1})$. For every idempotent e of the monoid, $N = eM_e e$ is another monoid where for all of its idempotents f, $eN_f e = e$. This is a property of the monoid, and therefore there is no distinct tier-based variant of this class.

Theorem 11. *If L is a* GLT *language, then L is definable in* $\textsc{fo}^2[<, \mathrm{bet}]$.

Proof. Let L be a GLT language and let M be its syntactic monoid. Then for any idempotent e, $eM_e e$ is a semilattice. For any x, y, and z in $eM_e e$, we have that $(xyz)^\omega y(xyz)^\omega = (xyz)y(xyz)$ by the idempotent property. Then by commutativity this is equivalent to $xxyyyzz$, and by the idempotent property again this reduces to xyz. Thus $(xyz)^\omega y(xyz)^\omega = (xyz)^\omega$, and $eM_e e$ belongs to $\mathbb{D}\mathbf{A} = \mathbf{M_e 1}$. ■

5.3.11 Languages Locally $\mathscr{J}$-Trivial

The piecewise-locally testable (PLT) languages of Rogers and Lambert (2019a) are characterized language-theoretically and logically. Logically, they are all and only those languages definable by Boolean combinations of factors that include both the successor and general precedence relations. Language-theoretically, if a language L is PLT, $w \in L$, and v has the same set of $\langle j, k\rangle$-factors as w, then $v \in L$ as well. A $\langle j, k\rangle$-factor is a collection of up to j substrings of length up to k, where the substrings are connected internally by $\lhd$ and each is before the one after it by $<$. Concretely, $\langle abcdef, \lhd < \lhd < \lhd\rangle$ is a $\langle 3, 2\rangle$-factor.

These factors and Boolean combinations thereof are clearly definable in $\textsc{fo}^2[<, \lhd]$, but there are there are languages in even just $\textsc{fo}^2[<]$ that are not PLT. Consider the language of all and only those words which contain an a that is eventually followed by a b which is itself never followed by c. This is $\text{FO}^2[<]$-definable:

$$(\exists x)[a(x) \wedge (\exists y)[y < x \wedge b(y) \wedge \neg(\exists x)[x < y \wedge c(x)]]].$$

The words $(b^{jk}a^{jk}c^{jk}a^{jk})^{j+1}$ and $(b^{jk}a^{jk}c^{jk}a^{jk})^{j}b^{jk}a^{jk}$ have the same $\langle j, k\rangle$-factors, but the former is rejected while the latter is accepted. Thus PLT is properly contained by $\text{FO}^2[<, \triangleleft]$. With multiplicity reducible to precedence, PLT subsumes LTT, but is incomparable with TSL· And of course PLT subsumes PT.

A class of languages exists with the same relationships to the other classes discussed here and which has not been shown to be equivalent to or distinct from PLT. Recall that the PT languages are all and only those whose syntactic monoids are $\mathscr{J}$-trivial. One might wonder then what the languages are whose semigroups satisfy this condition locally. This question has been explored before, for example by Knast (1983), and it has been shown that the class properly contains dot-depth one. Yet the $\mathscr{J}$-trivial monoids are all also in $\mathbb{D}\mathbf{A}$ (Brzozowski and Fich, 1984), so inclusion in $\text{FO}^2[<, \triangleleft]$ is guaranteed. The $\text{FO}^2[<]$ language where no a is eventually followed by a b which is itself never followed by c is, as expected, not in **LJ**, so incomparability with $\text{FO}^2[<]$ is confirmed. All PT languages are clearly in the class, however. Finally, recall that a language is locally threshold-testable iff for all idempotents e, f of its semigroup and all elements a, u, b, it holds that $eafuebf = ebfueaf$ and further this semigroup is aperiodic. We can then prove the following:

Theorem 12. *If L is locally threshold-testable, then its semigroup S is in* **LJ**.

Proof. Let L be locally threshold-testable and let S be its syntactic semigroup. Then suppose e is some idempotent of S and suppose $g \in eSe$. Then $h = g^w$ is an idempotent and we shall explore the submonoid $(eSe)_h$. First, note that eSe is commutative because $eae \cdot ebe = eaeeebe = ebeeeae = ebe \cdot eae$. Now let $u, v \in (eSe)_h$, implying that there exist elements a, b, c, and d such that $aub = h$ and $cvd = h$. Consider the following:

$$\begin{aligned} uhv &= uhhhv \\ &= u \cdot aub \cdot h \cdot cvd \cdot v \\ &= au^2bhcv^2d \\ &\dots \\ &= a^\omega u^\omega b^\omega hc^\omega v^\omega d^\omega. \end{aligned}$$

Because eSe is commutative, we can rearrange this to $(aub)^\omega h(cvd)^\omega = h^\omega hh^\omega = h$. Thus $uhv = h$ for all $u, v \in (eSe)_h$, which means that eSe is $\mathscr{J}$-trivial as desired. ■

It is unknown if the languages whose monoids lie in **LJ** are exactly the PLT languages, but if different the two classes have the same relationships with all of the other classes discussed so far.

This is a semigroup-based characterization, so there is a tier-based extension of the class. Notice that if the projected subsemigroup X of a language L is in **LJ**, and thus $\mathbf{Me}\mathbb{D}\mathbf{A}$ and star-free by containment, it must be the case that $X^1 = M(L)$ is in $\mathbf{Me}\mathbb{D}\mathbf{A}$. For all idempotents e of X^1, we have $1 \in X_e^1$ iff $e = 1$ by the star-free properties. Thus for all $e \neq 1$, $eX_e^1e = eX_ee$ and is in $\mathbb{D}\mathbf{A}$. And again by the star-free properties, $X_1^1 = \{1\} \in \mathbb{D}\mathbf{A}$. In other words, for all idempotents e of

X^1 we have that $eX^1_ee \in \mathbb{D}\mathbf{A}$. Thus all languages tier-based locally $\mathscr{J}$-trivial are definable in FO$^2[<, \mathrm{bet}]$, and this inclusion is proper: the language U_3 of Krebs et al. (2020), defined as the sublanguage of $\Sigma^*c(a+b)^*c\Sigma^*$ such that between the marked c no bb substring precedes an aa substring, is in the latter but not the former.

Note that the TLP class is incomparable with FO$^2[<, \lhd]$, as the language U_2 of Krebs et al. (2020) which forbids the occurrence of a bb substring anywhere before an aa substring over the $\{a, b\}$ tier is in the former but not the latter, while the language which forbids having an a eventually followed by a b which is not later followed by c (the language separating FO$^2[<]$ from $\mathbf{LJ}$) is in the latter but not the former.

5.3.12 Generalized Locally $\mathscr{J}$-Trivial Languages

In the same way that Brzozowski and Fich (1984) generalized the locally testable languages into the class here denoted GLT, one can generalize the locally $\mathscr{J}$-trivial languages into a new class here denoted GLP. The jump from requiring eSe to be a semilattice to requiring eM_ee to be a semilattice is the same jump from requiring eSe to be in $\mathbf{J}$ to requiring eM_ee to be in $\mathbf{J}$. This new class is of course a proper superclass of LP and TLP, with the language U_3 of Krebs et al. (2020) witnessing strictness of inclusion, and also a proper superclass of GLT, with U_3 again witnessing strictness. This class is incomparable with FO$^2[<, \lhd]$, with again U_3 not being in the latter, but with a language in the latter and outside this class defined as all and only those words w such that w contains an ab eventually followed by a cc that is itself never followed by another ab. This language then witnesses the strict inclusion of GLP by FO$^2[<, \mathrm{bet}]$.

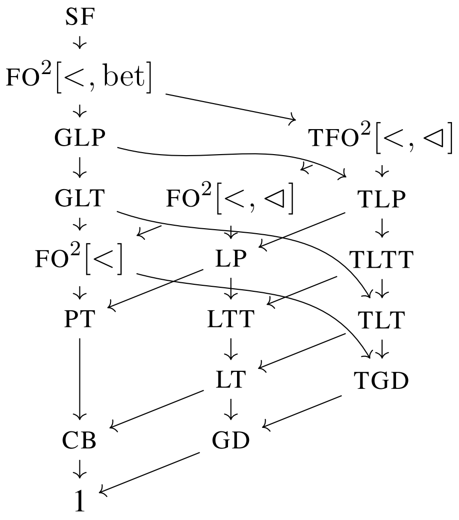


Figure 5.2: Hierarchy of inclusion for algebraically characterized classes. The locally $\mathscr{J}$-trivial class is denoted LP.

Table 5.1: A summary of classes and their characterizations.

	$\mathbf{V}$	$\mathbf{LV}$	$\mathbf{M_eV}$
$\mathbf{1}$	1	GD	$\mathrm{FO}^2[<]$
$\mathbf{J_I}$	CB	LT	GLT
$\mathbf{J}$	PT	LP	GLP
$\mathbb{D}\mathbf{A}$	$\mathrm{FO}^2[<]$	$\mathrm{FO}^2[<, \lhd]$	$\mathrm{FO}^2[<, \mathrm{bet}]$

5.4 Conclusions

Figure 5.2 shows all of the algebraically characterized classes of formal languages discussed in this chapter. An arrow from A to B indicates that A properly includes B, and there is a separating language available. If there is no path from A to B or from B to A, then the two are incomparable, and each contains a language not in the other. Table 5.1 offers a quick reference to the algebraic properties that characterize each of the classes. Each monoid variety $\mathbf{V}$ defines a class of languages, and two derived classes are formed: languages whose semigroups are locally in $\mathbf{V}$ (which admit a tier-based variant), and languages whose generated submonoids eM_ee are in $\mathbf{V}$ (which do not have a distinct tier-based variant).

The GLT, GLP, and $\text{FO}^2[<, \text{bet}]$ classes are each characterized by a monoid variety that admits further generalization by looking at those classes that are locally (or generalized locally) in that variety. Note that in Table 5.1 inclusion holds upward and leftward, and the rightmost column begins where the leftmost ends. Figure 5.2 will continue to spiral upward as the inclusions will continue to hold. It would be interesting to explore whether there is a fixed point, whether the spiral ends. If so, where would it be?

Chapter 6: Tree Recognition with Strongly Directed Hypergraphs

Finite-state acceptors over sequences are used for many purposes such as text search (Thompson, 1968) or control flow. They are commonly represented as a directed graph, but such an acceptor can represent only a regular language (Kleene, 1956). Using trees instead of sequences allows for exact classification of a context-free language, such as the Dyck language of balanced brackets, while maintaining a finite amount of state information (Rogers, 1997). Klein and Manning (2004) provide a hypergraph representation of finite-state acceptors over trees, but the resulting structure requires instantiating free indices in the nodes during a parse, which can result in the generation of unboundedly many ground states. This is unnecessarily redundant, generating a state-space which grows along with the input to be checked, mirroring a typical chart-parse.

This article introduces a generalization of hypergraphs that map sequences to sequences rather than mapping sets to sets. This can directly encode the transition relation of a tree acceptor, which maps a sequence of states to a single state. The resulting structure is a direct generalization of the graph structure used to represent string acceptors. Graph-based analyses of common decision problems and operations are provided. String acceptors can be reinterpreted unchanged as tree acceptors under this framework. When deciding if a tree is acceptable, the state-space remains constant no matter the size of the input. This can be useful pedagogically, as in contrast to introductory texts on string acceptors such as that of Hopcroft and Ullman (1979), works regarding tree acceptors such as those of Comon et al. (2007) or Gécseg and Steinby (1984) have no illustrations of the machines.

First we discuss background material in section 6.1. This includes the concepts of finite-state acceptors over sequences and over trees, as well as graph-based representations that have been used for each. Next, a generalization of the hypergraph is introduced in section 6.2, which will serve as a foundation for a novel representation of tree acceptors. Standard automaton operations including determinization, minimization, completion, trimming, and the Boolean operations are discussed in section 6.3. Finally we conclude with a summary.

6.1 Background

This section provides a brief overview of finite-state acceptors through a structural lens.

6.1.1 Trees

A sequence can be thought of as a linked list, with each position containing a symbol from the alphabet as well as a single connection to the rest of the structure. A tree is similar, except rather than a single connection there is an ordered sequence of connections to smaller structures. Traditionally trees are described in terms of a ranked alphabet, where the alphabet consists not of symbols alone, but of symbol/rank pairs. This bounds the expansion of the width of the tree. If the pair $\langle x, n \rangle$ appears in the alphabet, representing the symbol x with rank n, then a node labeled x is allowed to have exactly n children. A symbol may appear with more than one rank. That is, a tree is a structure built of the symbol x and a sequence of n subtrees.

Unranked tree languages, where there is no limit on the lengths of the sequences of subtrees, are used in some circumstances such as XML parsing (Barrero, 1991; Gelade et al., 2013). As will be discussed later when detailing completion and complementation algorithms, the structures described here can represent both ranked and unranked tree languages.

6.1.2 Finite-State Acceptors

Recall that a string acceptor is definable by a five-tuple $\langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is a finite alphabet, Q is a finite set of states, δ is a finite relation in $\Sigma \times Q \times Q$, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. Recall also that assignment of states may proceed either left-to-right or right-to-left, as regular languages are closed under reversal. When it comes to trees, the situation is similar. States may be assigned either top-down (from the root to the leaves) or bottom-up (from the leaves to the root). If δ may be an arbitrary relation, if the acceptor is nondeterministic, then both orderings are expressively equivalent (Comon et al., 2007). However if δ is required to be a function, if the acceptor is required to be deterministic, the bottom-up ordering is strictly more powerful than the top-down ordering, and indeed as expressive as a nondeterministic acceptor (Comon et al., 2007). Therefore only bottom-up acceptors will be considered here. A deterministic bottom-up finite-state tree acceptor for trees over a finite alphabet Σ can be defined by a finite set Q of states, a transition function $\delta\colon \Sigma \times Q^* \to Q$ whose preimage is finite, and a set $F \subseteq Q$ of accepting states. Note that the rank information can be inferred from the δ function and need not be explicitly encoded as part of Σ, although we will see in section 6.3 that the rank information is needed for completion. No initial state is needed, as the initial states are those reached by leaves (symbols of rank 0). If a node labeled x has children $\langle c_1, \ldots, c_n \rangle$ and each c_i has been assigned a state q_i, that node is assigned the state $\delta(x, \langle q_1, \ldots, q_n \rangle)$. This holds for leaves as well; a leaf labeled x is assigned the state $\delta(x, \varepsilon)$, where ε represents the empty sequence. This can be effectively computed by beginning at the root, descending to the leaves, and bubbling up the state information. Represent a tree by $x[t_1, \ldots, t_n]$, where x is the label of the root node, and each t_i is a child subtree. The state computed for this tree is $\delta^*(x[t_1, \ldots, t_n]) = \delta(x, \langle \delta^*(t_1), \ldots, \delta^*(t_n) \rangle)$. When a leaf is reached, the

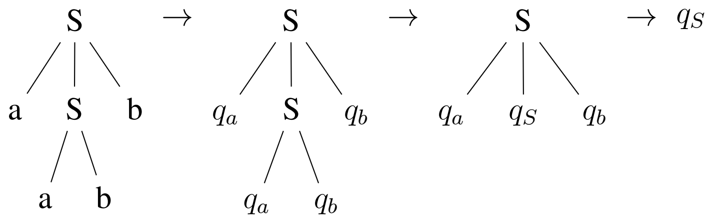


Figure 6.1: Accepting a tree.

sequence of children is empty and the base case is reached: $\delta^*(x[]) = \delta(x[])$.

Given a context-free grammar (CFG) defined by a set N of nonterminal symbols, a set T of terminal symbols, and a relation $R \subseteq N \times (N \cup T)^*$ describing rules, a deterministic bottom-up tree acceptors (DBFTA) representing the corresponding language can be constructed. For each terminal $t \in T$, assign $\delta(t, \varepsilon) = q_t$, and for each rule $n, \langle \sigma_1, \ldots, \sigma_n \rangle$, assign $\delta(n, \langle \sigma_1, \ldots, \sigma_n \rangle) = q_n$. This directly encodes the rules as a DBFTA, and all that remains is to mark the states associated with start symbols as accepting. Consider the CFG whose terminals are "a" and "b", whose sole nonterminal and start symbol is "S", and whose rules are S → ab and S → aSb. Figure 6.1 shows the assignment of states to a particular tree using the DBFTA derived from this CFG.

6.1.3 Directed Graphs and Extensions Thereof

A directed graph (a digraph) consists of a set V of nodes and a set $E \subseteq V \times V$ of edges between them. These edges may be labeled by elements of some finite alphabet Σ, in which case there is a labeling function $\lambda\colon \sigma \to \mathscr{P}(E)$. In other words, a labeled digraph is a collection of overlaid digraphs, sharing nodes but each having its own edges.[1] A finite-state string acceptor is representable as a labeled digraph, where the states are represented as nodes, the transition $\delta(\sigma, q_s) = q_f$ is represented

[1] Alternatively, a labeled graph may be represented as a heterogeneous 2-colored graph, where one color represents the actual nodes and the other represents the edge labels. The two are equivalent iff edges are allowed to lack sources or sinks.

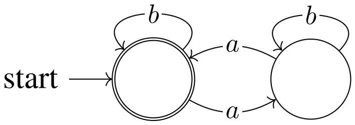


Figure 6.2: A finite-state string acceptor: doubly-outlined states are accepting, and the initial state is marked by "start". All and only those strings which contain an even number of occurrences of a are accepted.

by the existence of the edge $\langle q_s, q_f \rangle$ in $\lambda(\sigma)$, and the initial and accepting states are somehow marked as such. Figure 6.2 depicts a string acceptor whose associated set is all and only those strings which contain an even number of occurrences of the symbol 1 that has been discussed above.

If the set of edges were instead $E \subseteq \mathscr{P}(V) \times \mathscr{P}(V)$ then the underlying structure would be a hypergraph. This is not in itself sufficient to represent a tree acceptor, as trees are ordered. Moreover, a single state may inhabit two positions in the source set, which is not representable directly. That said, these hypergraph-structured automata have been used to represent trees by encoding free indices into the state-space (Klein and Manning, 2004). A representation of a tree acceptor as an unlabeled directed hypergraph like those of Klein and Manning (2004) is shown in Figure 6.3.

This representation is unsatisfying for at least two reasons. First, there is no ability to directly interpret a standard finite-state string acceptor as such a tree acceptor, despite the fact that a sequence is just a degenerate tree. Additionally, parsing a particular string instantiates the indices on the states, potentially many times. For instance, the string "aaabbb" will be parsed as follows:

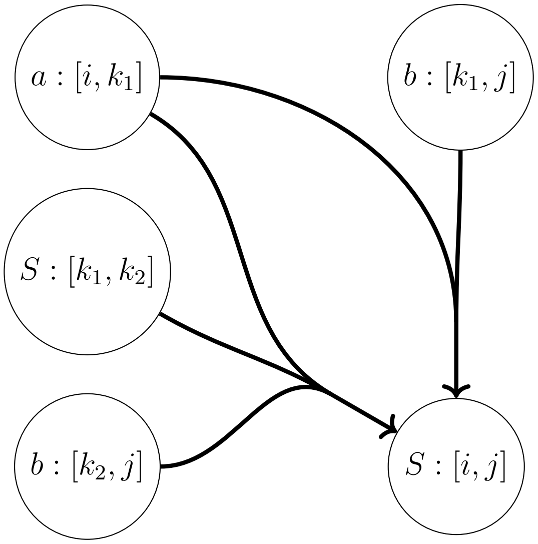
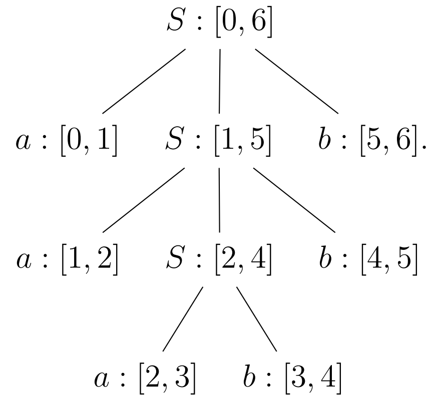


Figure 6.3: S → aSb and S → ab as an unlabeled directed hypergraph. Each rule is represented by a hyperedge.



Each node in this parse tree becomes a state in the derivation, despite the fact that $a : [i, k_1]$ appears only once in the automaton.

6.2 Strongly Directed Hypergraphs

This section introduces a variant of hypergraphs in which edges connect not sets to sets, but sequences to sequences. This modification avoids the need for multiple nodes per symbol, as well as multiple instantiations of nodes during parsing. Rather than having nodes for symbols augmented

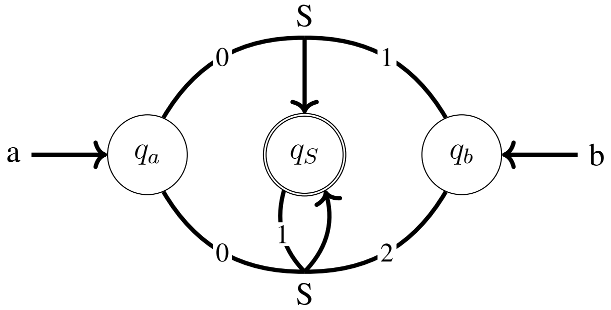


Figure 6.4: S → aSb and S → ab as a labeled strongly directed hypergraph. Each rule is represented by a hyperedge, and accepting states are represented by being doubly outlined.

by free indices, these objects will have one node per state of the acceptor, which for a context-free grammar need not be more than one per symbol.

A strongly directed hypergraph is a structure comprised of a set V of nodes and a set $E \subseteq V^* \times V^*$ of edges. Representing a DBFTA in this form is simple. As in the case of a string acceptor, each state is represented by one and only one node. A transition on symbol x from states $\langle q_1, \ldots, q_n \rangle$ to state q is represented by exactly such a labeled edge. The $\{S \rightarrow aSb, S \rightarrow ab\}$ tree acceptor is shown again in Figure 6.4, this time as a strongly directed hypergraph. Each edge has a sequence of sources and a sequence of sinks, denoted by numeric indices on the edges. In the case of a tree acceptor, the sequence of sinks is always singleton (they are B-graphs) and the redundant label is omitted.

An alternative graphical representation is inspired by Valdivia et al. (2021). Hypergraphs and their (strongly) directed variants can be represented in tabular format where each state labels a row and each edge labels a column. Accepting states are denoted by their row header being boxed. If a state is a source of an edge, the set of indices at which it appears is placed in the cell (or just some marker, if the sources are unordered). If a state is the sink of an edge, then the corresponding cell is marked.

	a	b	S	S
q_a	□		$\{0\}$	$\{0\}$
q_b		□	$\{1\}$	$\{2\}$
$\boxed{q_S}$			□	$\boxed{\{1\}}$

Figure 6.5: A tabular representation of the DBFTA of Figure 6.4.

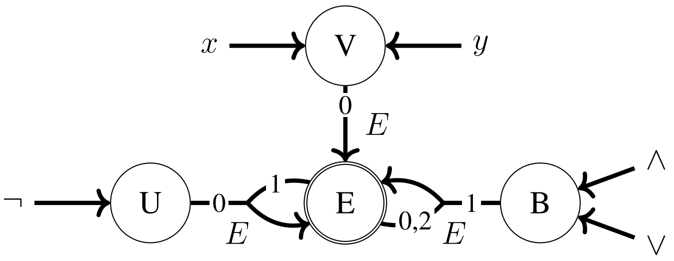


Figure 6.6: A tree acceptor for Boolean expressions over two variables.

The graphs in question here will never have multiple sinks, so denoting their indices is unnecessary. Figure 6.5 shows the same automaton as Figure 6.4. This tabular representation may be better for visualization purposes, since there are no layout constraints regarding the crossing of edges.

As another example, the following CFG, representing Boolean expressions over the two variables x and y, is represented in Figure 6.6 and Figure 6.7.

$$N = \{E\}$$

$$T = \{x, y, \neg, \wedge, \vee\}$$

$$R = \{E \to x, E \to y, E \to \neg E, E \to E \wedge E, E \to E \vee E\}$$

$$F = \{E\}$$

If both the source and sink sequences of edges are limited to singletons, this sort of tree acceptor is

	$\neg$	$\wedge$	$\vee$	x	y	E	E	E
U	□					$\{0\}$		
B		□	□				$\{1\}$	
V				□	□			$\{0\}$
E						$\boxed{\{1\}}$	$\boxed{\{0,2\}}$	□

Figure 6.7: The tabular representation of Figure 6.6.

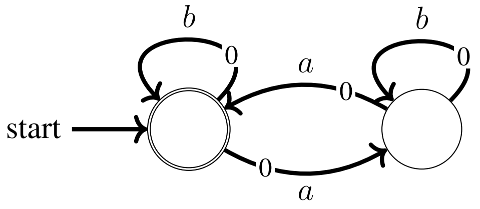


Figure 6.8: The string acceptor of Figure 6.2 as a strongly directed hypergraph. The two are identical, except that this version has additional labels for the sequence indices, which are always zero.

identical to a string acceptor. The string is then represented such that the rightmost symbol is the root and preceding symbols are children of their successors. A unique “start” symbol is added to occupy the leaf slot. This accounts for the fact that states are assigned from the bottom to the top.

6.3 Decisions and Operations

This section discusses decision procedures and automaton operations from the graph-based perspective. Concrete implementations are provided for some of the less trivial algorithms. Traditionally, tree automata operate over a ranked alphabet. The rank of a symbol can be inferred from the edges in the graph structure, although one should be careful to retain this information for symbol/rank pairs with no corresponding edges if the transition function is partial. All of these are straightforward generalizations of known algorithms for string acceptors, possibly after first reducing the strongly directed hypergraph to a simpler structure.

6.3.1 Reachability and Satisfiability

One of the questions that one may ask of a grammar is whether it is consistent, if there are any structures that satisfy it. The view of a tree acceptor as a strongly directed hypergraph yields an efficient test for finite-satisfiability of a given tree language L: there exists a tree in L iff there is some accepting state that is reachable. Reachability needs little information: all label information may be discarded, including both the sequence indices and the symbol labels. Then, a dynamic programming algorithm for deciding this question is as follows. If a node is the sink of an edge which has no sources, then that node is reachable. Once these are accounted for, one can iterate considering the following: if a node is the sink of an edge which has only reachable sources, then that node is reachable. Eventually, no updates will occur to the set of known-reachable nodes. At that point, everything not yet known to be reachable is unreachable. If any accepting state is reachable, then the DBFTA is satisfiable by a finite tree.

A DBFTA is reduced iff all of its states are reachable. To reduce a nonreduced DBFTA, simply remove all unreachable states as well as the edges to which they connect. Satisfiability is checking that the returned set of final states is nonempty.

6.3.2 Determinization

Sometimes one may be presented with a graph that represents not a deterministic function, but a nondeterministic relation. Because each edge has exactly one sink, the Rabin-Scott powerset construction (Rabin and Scott, 1959) applies to these tree automata in much the same way as to more typical string acceptors. Given a tree automaton with stateset Q and transition relation R, proceed as follows. For each symbol x which labels an edge with n sources[2] and for each possible sequence a of length n over $\mathscr{P}(Q)$, construct an edge as follows:

$$\delta(x, \langle a_1, \ldots, a_n \rangle) = \{q : (x, \langle q_1, \ldots, q_n \rangle, q) \in R \text{ and } q_i \in a_i\}.$$

Rather than constructing the entire powerset of states however, it is best to begin with the leaves and construct their corresponding states, then iteratively add edges considering only the states that exist so far. Eventually, no new states will be added, and a deterministic equivalent will have been constructed.

6.3.3 Minimization

Like Comon et al. (2007), this section describes DBFTA minimization in terms of the Myhill-Nerode theorem for trees. The following algorithm operates only on reduced deterministic automata. Similar to the string case, a table must be constructed with one fewer row and column than there are states. The table will be filled with partial derivations which distinguish the states. A first approximation of the partition, which will later be refined, says only that accepting states are distinct from rejecting states. Consider the DBFTA of Figure 6.9. It has four states, so the minimization table should have three rows and columns:

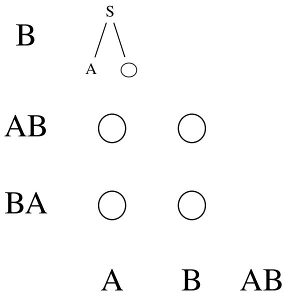
	A	B	AB
B			
AB	○	○	
BA	○	○	

[2] In other words, for each symbol x of rank n.

	a	b	S	S	S	S	S	S
A	□		0	1	0	0	2	2
B		□	1	0	2	2	0	0
$\boxed{\text{AB}}$			□		$\boxed{1}$	□	1	
$\boxed{\text{BA}}$				□		1	□	$\boxed{1}$

Figure 6.9: A nonminimal DBFTA. Set braces are omitted.

In this example, only two pairs of states remain undistinguished at this point. Next, to determine if states p and q are distinct, consider each symbol x that labels an edge with n sources. For each sequence a of length $n-1$ and for each $0 \leqslant i < n$, consider the outcome of the transition $\delta(x, \langle a_0, \ldots, a_{i-1}, p, a_i, \ldots, a_{n-1}\rangle)$ and that of the transition $\delta(x, \langle a_0, \ldots, a_{i-1}, q, a_i, \ldots, a_{n-1}\rangle)$. If the resulting states are known to be distinguishable, then p and q are also distinguishable. Here we see that states A and B are distinguished by $x = \text{S}$ and $a = \langle \text{A} \rangle$. Specifically $\delta(\text{S}, \langle \text{A}, \text{A}\rangle)$ is undefined (and thus rejecting) while $\delta(\text{S}, \langle \text{A}, \text{B}\rangle) = \text{AB}$ (and accepting). No such sequence yet suffices to distinguish AB from BA, however:



After another iteration, nothing changes. The states AB and BA are still not distinguishable, and no further updates can be made. All distinguished pairs are now known, and the partial trees stored in the table can help in constructing the distinguishing examples. If there is no need to find such an example, it suffices to simply store a mark in the cells representing distinguishable pairs.

	a	b	S	S	S	S
[A]	□		0	1	0	2
[B]		□	1	0	2	0
$\boxed{\text{[AB]}}$			□	□	$\boxed{1}$	$\boxed{1}$

Figure 6.10: A minimal form of Figure 6.9.

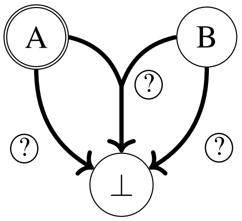
Minimization involves merging these indistinguishable states. For each state q_i, construct a state $[q_i]$ labeled by the set of states equivalent to q_i itself. Then for each edge, replace each state q in its sequence of sources and of sinks by its equivalence class. In this example, $[\text{AB}] = [\text{BA}] = \{\text{AB}, \text{BA}\}$, and all other classes are singleton. The minimal acceptor then is shown in Figure 6.10.

6.3.4 Completion and Trimming

A DBFTA is complete iff every n-source edge is attached to every possible sequence of n edges. A DBFTA is trim iff every state has some usable path to an accepting state. A path is usable iff every source of every edge along the path is reachable. To complete an automaton whose stateset is Q, find all symbols x that label edges of n symbols, and find all sequences a of length n over $Q \cup \{\perp\}$, where $\perp$ is some new state. Construct a new transition function

$$\delta'(x, a) = \begin{cases} \delta(x, a) & \text{if it is defined,} \\ \perp & \text{otherwise.} \end{cases}$$

If the automaton was already complete, then $\perp$ will be unreachable. Reduce the resulting automaton so that $\perp$ is not inserted unnecessarily. One may note that this only completes the automaton with respect to the supplied ranked alphabet. Several trees still have no path in the resulting automaton. In the case of string acceptors, one may work around this by adding edges labeled by some special



↦

	?⃝	?⃝	?⃝	S	S	S	S	S	S	S	S
$\boxed{\text{A}}$	?		?		0	1	2	01	02	12	012
B		?	?	012	12	02	01	2	1	0	
⊥	□	□	□	□	□	□	□	□	□	□	□

Figure 6.11: Instantiating ?⃝ for a rank-3 S.

symbol, ?⃝, which represents all symbols not in the alphabet (Hulden, 2009; Lambert and Rogers, 2020). The same applies in the case of tree acceptors, but there are significantly more of them. In order to account for universal completion, an edge on ?⃝ should be attached from every set of states to ⊥, including the empty set. Semantically adding a symbol/rank pair $\langle x, r\rangle$ to the alphabet requires duplicating and instantiating all such edges whose source sets have size bounded below by 1 and above by r to every possible surjective map from the first r natural numbers onto this source set. An example is shown in Figure 6.11.

The simplest way to trim a DBFTA is to first minimize and reduce it. All states lacking a path to an accepting state will be merged into a single state, as nothing can distinguish them. If a state lacks a path to an accepting state, then all of its usable out-edges leads to another such state. Because the DBFTA is minimal, that means these are self-loops. Thus trimming a minimal, reduced DBFTA involves inspecting the rejecting states in search of one where all it is the sink of all its out-edges. Then remove that state and all edges attached to it.

6.3.5 Finiteness

The language of a reduced, trimmed automaton is finite iff there are no cycles. Similar to the test for emptiness, the finiteness decision problem can be checked using a structure with reduced information capacity. Only a standard digraph is needed. Given the strongly directed hypergraph

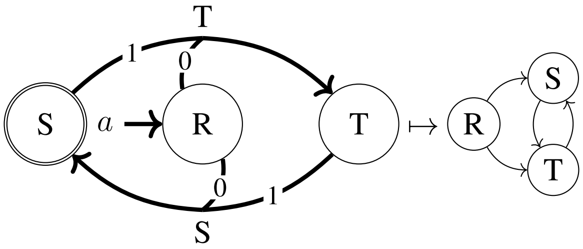


Figure 6.12: A DBFTA and its associated connection graph.

representation of a reduced, trimmed DBFTA, construct a connection graph as follows. Remove all state parity information. Replace each hyperedge of n sources with n edges, connecting each source to the sink. The resulting graph has cycles iff the original structure did as well. Reducedness is required to guarantee that all edges are traversable, and trimness removes the rejecting sink and its associated loops. Figure 6.12 shows a DBFTA and its connection graph, each of which has a clearly visible cycle of length two. The advantage of using the connection graph is that cycle search in graphs is already commonly implemented.

6.3.6 Boolean Operations

The complement of a complete DBFTA is found by swapping the parity of each state. All rejecting states become accepting, and all accepting states become rejecting. Note that this is really only a complement relative to the set of all trees over the specified ranked alphabet, and that there exist trees that are in neither the represented tree language nor its complement due to containing symbol/rank pairs outside of this alphabet. The same caveat applies to string acceptors and the same workaround is useful: completion with ⓘ edges. Barrero (1991) proves that tree acceptors cannot represent the complement of an unranked tree language, but the ⓘ edges essentially act as infinitely many edges, bypassing this restriction.

The union of two tree automata is even simpler than the complement. The simplest construction is the same as for string acceptors. If each is represented by a graph, then both graphs together, with no connections at all between them, suffice to give a nondeterministic representation of their union. If either has edges on $\textcircled{?}$, they should be instantiated as necessary such that both graphs operate over the same ranked alphabet. The resulting disconnected graph can be determinized, minimized, and trimmed if desired, but as noted by Heinz and Rogers (2013) it may often be preferable to leave Boolean combinations in this factored state. The simplest implementation of intersection involves application of De Morgan's laws: $A \cap B = \complement(\complement A \cup \complement B)$. Unfortunately, complementation requires determinism, but the factored union representation is nondeterministic. So there is no intrinsic factored representation of intersections.

Alternatively, the union and intersection can be represented by the product construction. Provided are two tree acceptors $\mathcal{A}_i = \langle \Sigma_i, Q_i, \delta_i, F_i \rangle$ for $i \in \{1, 2\}$. For each symbol x of rank zero, construct a state $\langle q_a, q_b \rangle$, where $q_a = \delta_1(x, \varepsilon)$ and $q_b = \delta_2(x, \varepsilon)$. Construct an edge from ε to $\langle q_a, q_b \rangle$. Then iterate as follows. Let Q represent the set of states generated so far. Then for each symbol/rank pair $\langle x, r \rangle$ and for each sequence s of length r over Q, let $\langle u_i, v_i \rangle = s_i$ and construct a state $\langle q_a, q_b \rangle$ where $q_a = \delta_1(x, u)$ and $q_b = \delta_b(x, v)$. Construct an edge from s to $\langle q_a, q_b \rangle$, and repeat. Eventually no new states will have been created, and the necessary portion of the product is generated. For the union, the accepting states are those where either component of the labeling pair is accepting in its associated acceptor. For the intersection, both components must be accepting. Continuing a recurring theme, this is the same algorithm as for string acceptors, except that it has to account for all possible sequences of sources rather than considering only a single state at a time.

6.4 Conclusions

This chapter introduced a novel generalization of hypergraphs which allows a direct encoding of a finite-state tree acceptor in a graph-like structure. Decision problems and Boolean operations are interpreted with respect to these structures, in some cases by reducing them to simpler structures. The meaning of complementation was discussed, and a solution implemented to guarantee that all possible trees exist are accepted by either a DBFTA or its complement. The framework allows for reinterpretation of finite-state string acceptors as tree acceptors without change.

Chapter 7: Accumulators and the Problems They Bring

Some have expressed interest in describing certain phenomena using semirings (Lothaire, 2005). These are essentially finite-state acceptors augmented with a monoidal accumulator. This chapter discusses the expressive power of such a structure. First I show that a CKY-style chart parse can be expressed as a monoid. Then I generalize, demonstrating that a run of a Turing machine can also be expressed as a monoid. In order to maintain a truly finite amount of state, one may demand the use of a finite monoid, but this prohibits representing even the identity function. A fundamental question remains: what kinds of restrictions should a monoidal accumulator satisfy?

7.1 Parsing as a Monoid

Originally published by Ichirō Sakai (Sakai, 1962) and later rediscovered by John Cocke (Hays, 1962), Tadao Kasami (Kasami, 1966), and Daniel Younger (Younger, 1967), the CKY algorithm (named for the latter trio) is a cubic-time bottom-up dynamic programming parser for context-free grammars. This section describes the algorithm in terms of a monoid.

A **context-free grammar** (a type 2 phrase-structure grammar) is a system of rewrite rules, consisting of a set of terminal symbols (words), nonterminal symbols, and rules that transform nonterminal symbols into a sequence of zero or more symbols of either variety (Chomsky, 1956, 1959). They are called "context-free" because a type 1 ("context-sensitive") grammar allows rules to apply only in given contexts.

For notation here, nonterminal symbols are represented by (sequences of) capital letters and terminals by anything else. A rule is written $A \rightarrow w$, where A is the nonterminal being rewritten and w is the output sequence.

Consider the grammar with terminals '(' and ')', nonterminal S, and rules S $\rightarrow$ (), S $\rightarrow$ SS, and S $\rightarrow$ (S). It generates the Dyck language of balanced parentheses such as "()", "()()", "(())", etc. This is equivalent (in terms of string yield) to a grammar with the same terminals but a larger set of nonterminals, $\{\mathrm{S}, \mathrm{T}, \mathrm{L}, \mathrm{R}\}$, and the following set of rules: S $\rightarrow$ LR, S $\rightarrow$ SS, S $\rightarrow$ LT, T $\rightarrow$ SR, L $\rightarrow$ (, and R $\rightarrow$). Figure 7.1 shows some sample parses with this latter grammar. It has a few features that make processing easier.

This second grammar is in a special form known as **Chomsky Normal Form**, where a nonterminal rewrites to either a single terminal or a sequence of exactly two nonterminals. Any context-free grammar can be written in this form. This arrangement lends itself nicely to a divide-and-conquer style of algorithm. The grammar used in subsequent examples is:

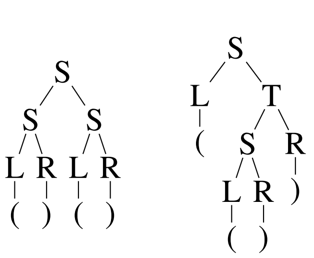


Figure 7.1: Parses for "()()" and "(())".

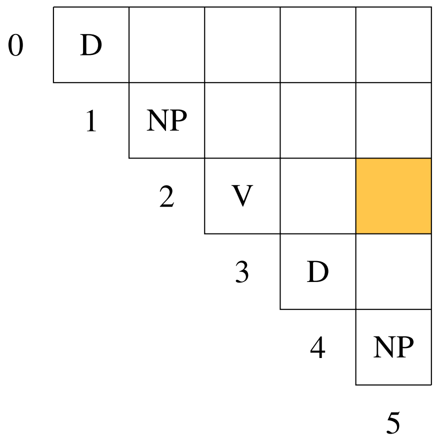


Figure 7.2: Basic shape of the parsing matrix for “$_0$the$_1$girl[$_2$saw$_3$the$_4$crow$_5$]”. The cell in row 2, column 5 is highlighted and corresponds to the bracketed region of the input from index 2 to 5.

S → DP VP	D → the
DP → DP PP	NP → binoculars
DP → D NP	NP → crow
PP → P DP	NP → girl
VP → VP PP	P → with
VP → V DP	V → saw

For a sentence of n words, parsing occurs in an upper-triangular n-dimensional matrix whose rows are numbered from zero and whose columns are numbered from one. Concretely, consider the sentence “the girl saw the crow”. Indices are placed between words, with 0 at the start of the sentence and n after word n. Figure 7.2 shows the structure of the matrix.

The cells along the main diagonal correspond to single words. If a nonterminal N can rewrite to that word w, then a tree $N \to w$ is placed in that cell. These have also been filled in (denoted only by N) in Fig. 7.2.

0	D	DP			S
	1	NP			
		2	V		VP
			3	D	DP
				4	NP
					5

Figure 7.3: Parsed “$_0$the$_1$girl$_2$saw$_3$the$_4$crow$_5$”.

For any other cell, the span from i to k that it represents can be split in a number of ways. For all j where $i < j < k$, it can be split into a region spanning i to j and one spanning j to k. The highlighted cell spanning 2 through 5 in Fig. 7.2 can split into 2–3 and 3–5, or into 2–4 and 4–5. For each tree T_1 in the cell corresponding to the left region and for each T_2 in the right region, the roots of T_1 and T_2 will be nonterminals N_1 and N_2. If there is a rule rewriting N to $N_1 N_2$ for some nonterminal N, then a tree $N \rightarrow N_1, N_2$ is placed in that cell. If no such rules exist, then the cell must remain empty. Figure 7.3 shows the entire table filled.

Cells are not limited to a single entry. Some sentences offer more than one valid parse, and the result will be a forest of possible interpretations. Consider the sentence “the girl saw the crow with the binoculars”, shown parsed in Figure 7.4. After having parsed the three additional words into their own structure, their matrix can be adjoined diagonally to the earlier structure. Only for the newly formed cells where the rows from the left portion and columns from the right meet, highlighted in the figure, is there work to be done. There are two instances of VP in the cell spanning 2–8, because both the 2–3/3–8 (V DP) split, representing the case where the crow has binoculars, and the 2–5/5–8 (VP PP) split, where the girl has binoculars, are valid interpretations of the sentence. For the same

0	D	DP			S			S/S
	1	NP						
		2	V		VP			VP/VP
			3	D	DP			DP
				4	NP			
					5	P		PP
						6	D	DP
							7	NP
								8

Figure 7.4: Parsed “$_0$the$_1$girl$_2$saw$_3$the$_4$crow$_5$with$_6$the$_7$binoculars$_8$”.

reason, there are two instances of S in the upper right cell. Note that the S from 0–5 no longer represents a valid parse of the sentence, as this span does not contain the entirety of the sentence. We can see then that any given CFG has a parsing monoid associated with it: these upper triangular matrices are the elements, and the operation is the act of combining them as was done here.

The preceding portion describes the use of a standard grammar. Weighted or stochastic grammars may be used with only a slight modification. Along the main diagonal, the weight assigned to the tree is exactly the weight of the rule that generated it. In the other cells, the weight is an appropriate combination of the weights of the subtrees with that of the generating rule. This combination might be a product (Smith and Johnson, 2007) or a sum (Katsirelo et al., 2008). The parsing monoid is formed exactly as before, with the elements being the matrices and the operation being the act of diagonally adjoining them then filling in the newly created cells.

A finite-state machine with a single state and a monoidal accumulator is thus sufficient to parse context-free languages. Each input symbol loops back to this same state while outputting a single-cell parse chart, and the accumulator dutifully builds the table as per the monoid operation. One may object that the size of the structure within the accumulator grows quadratically with the size of the input, and this complaint would be well-founded. The infinitely many states needed to decide well-formedness of the context-free string language have been pushed into the monoid.

7.2 Going Further: Turing Completeness

Recall that a monoid is a set S equipped with an associative total function $\Diamond\colon S \times S \to S$, where some element $e \in S$ is an identity for $\Diamond$, i.e. for all $x \in S$, it holds that $e \Diamond x = x = x \Diamond e$. Given a function $f\colon B \times A \to B$, an initial value b, and a sequence σ of elements of A, the function that computes the following:

$$f(\dots f(f(b, \sigma_1), \sigma_2) \dots, \sigma_n)$$

is a left-fold. Here I show that for any such f, a new function $\square$ can be constructed such that $\square$ is a monoid operation.

Suppose we are given a function $f\colon B \times A \to B$. The two types need not be the same, so it is clear that the function need not be associative. Consider new unit types L, N, and R and construct a new aggregate type:

$$C = \big(L \times B\big) \cup N \cup \big(R \times A\big).$$

We can then define a function $g\colon C \to C$:

$$g(x, y) = \begin{cases} \langle L, f(b, a) \rangle & \text{if } x = \langle L, b \rangle,\, y = \langle R, a \rangle \\ y & \text{if } x = N \\ x & \text{otherwise.} \end{cases}$$

Then we have that $g(N, c) = c = g(c, N)$, or in other words N is an identity for g. Then we have a set C and a binary operation over that set with an identity. The only thing missing is a guarantee of associativity. But no matter. Let us finally define yet another function $\square\colon ([C] \times C) \times ([C] \times C) \to$

$([C] \times C)$ as follows:

$$a \,\square\, b = \begin{cases} b & \text{if } a = \langle \varnothing, N \rangle \\ a & \text{if } b = \langle \varnothing, N \rangle \\ \langle x + y, & \\ \quad \text{foldl}(g, N, x + y) \rangle & a = \langle x, c \rangle, \text{b=}\langle y, d \rangle. \end{cases}$$

The pair $\langle \varnothing, N \rangle$ is an identity for $\square$. Then for both of $a \,\square (b \,\square\, c)$ and $(a \,\square\, b) \,\square\, c$, the first element of the resulting tuple (which represents an ordered list of seen elements) is identical, and the other element is computed from that alone. So we have a set $([C] \times C)$ with an associative total binary operation ($\square$) and an identity ($\langle \varnothing, N \rangle$). Thus we have a monoid.

Then an automaton with a single monoid-type accumulator is enough to compute any fold: there is a single state whose start-edge is labelled with a pair of the form $\langle [\langle L, i \rangle], \langle L, i \rangle \rangle$ and all other edges (and the termination output) are labelled with either $\langle \varnothing, N \rangle$ or pairs of the form $\langle [\langle R, x \rangle], \langle R, x \rangle \rangle$. The monoid operation is of course the $\square$ derived from the intended fold function. Computing right-folds instead of left-folds requires only minor modification.

Thus a finite-state machine with a monoidal accumulator and just a single state can express any fold. Just like with parsing, all of the state information is pushed into the monoid. And it should be noted that an infinite monoid is generated in the process which enforces associativity, remembering the entire input as it progresses. A run of a Turing machine can be expressed in this way: collect the input, and run the intended Turing machine over the collected input. Restricting ourselves to finite monoids certainly reduces the computational power, as the input can no longer be remembered. Indeed, Σ^* is an infinite monoid, so even the identity transformation is no longer representable under such a harsh restriction.

7.3 Conclusions

The semiring-based analysis of transduction by Lothaire (2005) is elegant in its simplicity, but when unrestrained it produces unlimited computational power (Hutton, 1999). A problem arises when trying to appropriately restrict the power. This analysis was designed to unify acceptors and transducers, but a restriction to finite monoids prevents defining even the most trivial of transductions. Perhaps the types of monoids allowed should be all and only those whose operations are computable with a finite-state machine, but this would include these machines, so we have to be careful not to accidentally allow for, say, all primitive recursive functions.

Chapter 8: Conclusions

I presented here a structure unifying between string acceptors and tree acceptors in order to improve our understanding of linguistic structure. For the same reasons, I have discussed characterizations and learning algorithms for the tier-based extensions of classes of formal languages in the piecewise-local subregular hierarchy, and additionally situated into this hierarchy some classes based on definability in restrictions of first-order logic. Further, I have discussed the downfalls of using algebraic structures for transducers without regard to their expressive power.

The characterizations of the classes of the piecewise-local subregular hierarchy provide information regarding the properties of the languages they contain. Language-theoretic characterizations allow one to prove that a given formal language lies outside of a class, that the properties may not hold. Of course, such a characterization can also prove inclusion if the entire language is accessible, but a negative answer requires only finitely many words witnessing an exceptions. Model-theoretic results describe the kind of logic required in order to be able to represent a class. These can easily provide a positive guarantee that a language is in the class, but it may be more difficult to show that a language cannot be represented in the given form. Algebraic characterizations based on syntactic semigroups or syntactic monoids can provide both positive and negative inclusion claims. With these in mind, I characterized the tier-based extensions of the classes of the piecewise-local subregular hierarchy language-theoretically, model-theoretically, and algebraically. Because the complements of the strictly local languages had not formerly been characterized language-theoretically, I include those results as well. Having thoroughly explored the tier-based extensions of formal language classes, I go on to provide online learning algorithms for them as well. However, learning of the tier of salient symbols is a constant plague for these classes. Learning the multiple-tier-based classes can bypass this issue, but is unfeasible.

Beyond the tier-based extensions of subregular classes, several classes of formal languages are known from the computer science literature which have not, to my knowledge, been used with respect to linguistic description. These are the classes based on definability in first-order logic restricted to two variables. I show how these classes fit in with the others in the subregular hierarchy by using the appropriate formalism to define factors over the commonly used relations or by finding languages that fail to satisfy the algebraic characterizations of the classes.

String languages and tree languages have traditionally been treated in different ways. While the algebraic theory of trees has not been completely settled, I do provide a unifying structure for string acceptors and tree acceptors, such that operations on tree languages are simplified and perhaps an algebraic theory might progress via the same route as for strings.

The leading algebraic theory of finite-state functions, that based on semirings from Lothaire (2005), offers a beautiful unification of string acceptors and string transducers. It can define standard acceptors, weighted automata, string-to-string relations both with and without probabilities, and more. So much more. With particular choices of semiring, parsing can be represented with a machine of a single state. In fact, any computable function can be represented in this way. All of the state is pushed into the semiring. But when restricted to finite structures, transformations become impossible and the unifying beauty of the formalism is lost.

In short, I hope that discussion of a structure-based approach to linguistic analysis may encourage others to more freely discuss the variety of ways in which natural language patterns can be described and to consider structural and learnability concerns when proposing new classes of formal languages and transformations for linguistic purposes.

Paths for future exploration are numerous. Mixed relation functions, like the piecewise-locally testable and strictly piecewise-local classes defined by factors involving both adjacency and general precedence, are still largely unexplored. Perhaps there are varieties of monoids or semigroups to which these correspond, or perhaps the algebraic view might expose a similar class worth exploring. Further, syntax relies on trees, and while there are some ideas regarding a subregular hierarchy of tree languages, it is nowhere neare as well-developed as that for strings. And finally, exploring subregular (or, subrational) classes of relations may prove useful in describing phonological or morphological transformations.

Bibliography

Alfred Vaino Aho, Michael Randolph Garey, and Jeffrey David Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. doi: 10.1137/0201008.

Alëna Aksënova and Sanket Deshmukh. Formal restrictions on multiple tiers. In *Proceedings of the Society for Computation in Linguistics*, volume 1, pages 64–73, Salt Lake City, Utah, 2018. doi: 10.7275/R5K64G8S.

Alejandro Barrero. Unranked tree languages. *Pattern Recognition*, 24(1):9–18, 1991. doi: 10.1016/0031-3203(91)90112-I.

Danièle Beauquier and Jean-Éric Pin. 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, 1989. doi: 10.1007/BFb0035752.

Danièle Beauquier and Jean-Éric Pin. Languages and scanners. *Theoretical Computer Science*, 84 (1):3–21, July 1991. doi: 10.1016/0304-3975(91)90258-4.

Kenneth Reid Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI Publications, 2003.

Mikołaj Bojańczyk. Transducers with origin information. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8–11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer Berlin / Heidelberg, 2014. doi: 10.1007/978-3-662-43951-7_3.

Mikołaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which classes of origin graphs are generated by transducers? In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 114:1–114:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi: 10.4230/LIPIcs.ICALP.2017.114.

Janusz Antoni Brzozowski and Faith Ellen Fich. On generalized locally testable languages. *Discrete Mathematics*, 50:153–169, 1984. doi: 10.1016/0012-365X(84)90045-1.

Janusz Antoni Brzozowski and Imre Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, March 1973. doi: 10.1016/S0012-365X(73)80005-6.

Julius Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6(1–6):66–92, 1960. doi: 10.1002/malq.19600060105.

Phillip Burness and Kevin McMullin. Efficient learning of output tier-based strictly 2-local functions. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 78–90, Toronto, Canada, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5707.

Pascal Caron. LANGAGE: A Maple package for automaton characterization of regular languages. In Derick Wood and Sheng Yu, editors, *Automata Implementation*, volume 1436 of *Lecture Notes in Computer Science*, pages 46–55. Springer Berlin / Heidelberg, 1998. doi: 10.1007/BFb0031380.

Jane Chandlee. *Strictly Local Phonological Processes*. PhD thesis, University of Delaware, 2014. URL `https://chandlee.sites.haverford.edu/wp-content/uploads/2015/05/Chandlee_dissertation_2014.pdf`.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, September 1956. doi: 10.1109/TIT.1956.1056813.

Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, June 1959. doi: 10.1016/S0019-9958(59)90362-6.

Alexander Clark. The syntactic concept lattice: Another algebraic theory of the context-free languages? *Journal of Logic and Computation*, 25(5):1203–1229, October 2015. doi: 10.1093/logcom/ext037.

Robin Clark and Ian Roberts. A computational model of language learnability and language change. *Linguistic Inquiry*, 24(2):299–345, 1993.

Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications, October 2007. URL `http://tata.gforge.inria.fr`.

Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126(1):53–75, April 1994. doi: 10.1016/0304-3975(94)90268-2.

András Cser. The -alis/-aris allomorphy revisited. In Franz Rainer, Wolfgang Dressler, Dieter Kastovsky, and Hans Christian Luschützky, editors, *Variation and Change in Morphology: Selected Papers from the 13th International Morphology Meeting*, pages 33–52. John Benjamins Publishing Company, Vienna, Austria, 2010. doi: 10.1075/cilt.310.02cse.

Aldo De Luca and Antonio Restivo. A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup. *Information and Control*, 44(3):300–319, March 1980. doi: 10.1016/S0019-9958(80)90180-1.

Aniello De Santo and Thomas Graf. Structure sensitive tier projection: Applications and formal properties. In Raffaella Bernardi, Greg Kobele, and Sylvain Pogodalla, editors, *Formal Grammar 2019*, volume 11668 of *Lecture Notes in Computer Science*, pages 35–50. Springer Verlag, 2019. doi: 10.1007/978-3-662-59648-7_3.

Hossep Dolatian and Jonathan Rawski. Multi-input strictly local functions for templatic morphology. In *Proceedings of the Society for Computation in Linguistics*, volume 3, pages 282–296, New Orleans, Louisiana, 2020. URL `https://scholarworks.umass.edu/scil/vol3/iss1/28`.

Matt Edlefsen, Dylan Leeman, Nathan Myers, Nathaniel Smith, Molly Visscher, and David Wellcome. Deciding strictly local (SL) languages. In Jon Breitenbucher, editor, *Proceedings of the 2008 Midstates Conference for Undergraduate Research in Computer Science and Mathematics*, pages 66–73, 2008.

Samuel Eilenberg and Marcel-Paul Schützenberger. On pseudovarieties. *Advances in Mathematics*, 19(3):413–418, March 1976. doi: 10.1016/0001-8708(76)90029-3.

Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, January 1961. doi: 10.2307/1993511.

Faith Ellen Fich and Janusz Antoni Brzozowski. A characterization of a dot-depth two analogue of generalized definite languages. In Hermann A. Maurer, editor, *Automata, Languages and Programming. ICALP 1979*, volume 71 of *Lecture Notes in Computer Science*, pages 230–244. Springer-Verlag, July 1979. doi: 10.1007/3-540-09510-1_18.

Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108:45–82, 1993. doi: 10.1016/0304-3975(93)90230-Q.

Jie Fu, Jeffrey Heinz, and Herbert G. Tanner. An algebraic characterization of strictly piecewise languages. In Mitsunori Ogihara and Jun Tarui, editors, *Theory and Applications of Models of Computation*, volume 6648 of *Lecture Notes in Computer Science*, pages 252–263. Springer Berlin / Heidelberg, 2011. doi: 10.1007/978-3-642-20877-5_26.

Pedro Garcia, Enrique Vidal, and José Oncina. Learning locally testable languages in the strict sense. In *Proceedings of the 1st International Workshop on Algorithmic Learning Theory*, pages 325–338, Tokyo, Japan, 1990. URL `https://grfia.dlsi.ua.es/repositori/grfia/pubs/111/alt1990.pdf`.

Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, Hungary, 1984.

Wouter Gelade, Tomasz Idziaszek, Wim Martens, Frank Neven, and Jan Paredaens. Simplifying XML schema: Single-type approximations of regular tree languages. *Journal of Computer and System Sciences*, 79(6):910–936, September 2013. doi: 10.1016/j.jcss.2013.01.009.

Christian Germain and Jean Pallo. Langages rationnels définis avec une concaténation non-associative. *Theoretical Computer Science*, 233(1–2):217–231, February 2000. doi: 10.1016/S0304-3975(98)00043-7.

R. W. N. Goedemans, Jeffrey Heinz, and Harry van der Hulst. StressTyp2, April 2015. URL `http://st2.ullet.net/`.

Edward Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, May 1967. doi: 10.1016/S0019-9958(67)91165-5.

Erich Grädel and Martin Otto. On logics with two variables. *Theoretical Computer Science*, 224 (1–2):73–113, August 1999. doi: 10.1016/S0304-3975(98)00308-9.

Thomas Graf. The power of locality domains in phonology. *Phonology*, 34(2):385–405, 2017. doi: 10.1017/S0952675717000197.

Thomas Graf. Why movement comes for free once you have ajdunction. In Daniel Edmiston, Marina Ermolaeva, Emre Hakgüder, Jackie Lai, Kathryn Montemurro, Brandon Rhodes, Amara Sankhagowit, and Michael Tabatowski, editors, *Proceedings of the Fifty-third Annual Meeting of the Chicago Linguistic Society*, pages 117–136, Chicago, Illinois, 2018. Chicago Linguistic Society.

James Alexander Green. On the structure of semigroups. *Annals of Mathematics*, 54(1):163–172, July 1951. doi: 10.2307/1969317.

Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969. doi: 10.1016/s0021-9800(69)80111-0.

David Glenn Hays. Automatic language-data processing. In Harold Borko, editor, *Computer Applications in the Behavioral Sciences*, pages 394–423. 1962.

Jeffrey Heinz. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661, October 2010a. doi: 10.1162/ling_a_00015.

Jeffrey Heinz. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden, July 2010b. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P10-1092`.

Jeffrey Heinz and James Rogers. Learning subregular classes of languages with factored deterministic automata. In *Proceedings of the 13th Meeting on the Mathematics of Language*, pages 64–71, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W13-3007`.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 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, 2011. Association for Computational Linguistics. URL `https://aclweb.org/anthology/P11-2011`.

Jeffrey Heinz, Anna Kasprzik, and Timo Kötzing. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127, October 2012. doi: 10.1016/j.tcs.2012.07.017.

Markus Holzer and Barbara König. On deterministic finite automata and syntactic monoid size. *Theoretical Computer Science*, 327(3):319–347, November 2004. doi: 10.1016/j.tcs.2004.04.010.

John Edward Hopcroft and Jeffrey David Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

Liang Huang and David Chiang. Better k-best parsing. In Harry Bunt and Robert Malouf, editors, *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–65, Vancouver, British Columbia, October 2005. URL `https://aclanthology.org/W05-1506`.

Mans Hulden. *Finite-State Machine Construction Methods and Algorithms for Phonology and Morphology*. PhD thesis, The University of Arizona, 2009. URL `https://hdl.handle.net/10150/196112`.

Graham Hutton. A tutorial on the universality and expressiveness of fold. *Journal of Functional Programming*, 9(4):355–372, 1999. doi: 10.1017/s0956796899003500.

Larry M. Hyman and Francis X. Katamba. A new approach to tone in Luganda. *Language*, 69(1): 34–67, March 1993. doi: 10.2307/416415.

Sanjay Jain, Steffen Lange, and Sandra Zilles. Some natural conditions on incremental learning. *Information and Computation*, 205(11):1671–1684, November 2007. doi: 10.1016/j.ic.2007.06.002.

Adam Jardine and Jeffrey Heinz. Learning tier-based strictly 2-local languages. *Transactions of the Association for Computation in Linguistics*, 4:87–98, 2016. doi: 10.1162/tacl_a_00085.

Adam Jardine and Kevin McMullin. Efficient learning of tier-based strictly k-local languages. In Frank Drewes, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications: 11th International Conference*, volume 10168 of *Lecture Notes in Computer Science*, pages 64–76. Springer, Cham, 2017. doi: 10.1007/978-3-319-53733-7_4.

Jing Ji and Jeffrey Heinz. Input strictly local tree transducers. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications: Proceedings of the 14th International Conference, LATA 2020*, volume 12038 of *Theoretical Computer Science and General Issues*, pages 369–381, Cham, Switzerland, 2020. Springer International Publishing. doi: 10.1007/978-3-030-40608-0_26.

Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical Report R-257, University of Illinois, Urbana, Illinois, March 1966.

George Katsirelo, Nina Naraodytska, and Toby Walsh. The weighted CFG constraint. In Laurent Perron and Michael Alan Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2008*, volume 5015 of *Lecture Notes in Computer Science*, pages 323–327. Springer Berlin / Heidelberg, 2008. doi: 10.1007/978-3-540-68155-7_31.

Stephen Cole Kleene. Representation of events in nerve nets and finite automata. In Claude Elwood Shannon and John McCarthy, editors, *Automata Studies*, volume 34 of *Annals of Mathematics Studies*, pages 3–42. Princeton University Press, 1956. doi: 10.1515/9781400882618-002.

Dan Klein and Christopher D. Manning. Parsing and hypergraphs. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, volume 23 of *Text, Speech and Language Technology*, pages 351–372. Springer Dordrecht, 2004. doi: 10.1007/1-4020-2295-6_18.

Robert Knast. A semigroup characterization of dot-depth one languages. *RAIRO – Informatique théorique*, 17(4):321–330, 1983. doi: 10.1051/ita/1983170403211.

Donald Ervin Knuth, James Hiram Morris, and Vaughan Ronald Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, August 1977. doi: 10.1137/0206024.

Andreas Krebs, Kamal Lodaya, Paritosh K. Pandya, and Howard Straubing. Two-variable logics with some betweenness relations: Expressiveness, satisfiability, and membership. *Logical Methods in Computer Science*, 16(3):1–41, September 2020. doi: 10.23638/LMCS-16(3:16)2020.

Regine Lai. Learnable vs. unlearnable harmony patterns. *Linguistic Inquiry*, 46(3):425–451, July 2015. doi: 10.1162/LING_a_00188.

Dakotah Lambert. Grammar interpretations and learning TSL online. In *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 81–91, August 2021a. URL `https://proceedings.mlr.press/v153/lambert21a.html`.

Dakotah Lambert. Relativized adjacency. *Journal of Logic, Language and Information*, 2021b. Accepted with minor revisions.

Dakotah Lambert and James Rogers. A logical and computational methodology for exploring systems of phonotactic constraints. In *Proceedings of the Society for Computation in Linguistics*, volume 2, pages 247–256, New York City, New York, 2019. doi: 10.7275/t0dv-9t05.

Dakotah Lambert and James Rogers. Tier-based strictly local stringsets: Perspectives from model and automata theory. In *Proceedings of the Society for Computation in Linguistics*, volume 3, pages 330–337, New Orleans, Louisiana, 2020. doi: 10.7275/2n1j-pj39.

Dakotah Lambert, Jon Rawski, and Jeffrey Heinz. Typology emerges from simplicity in representations and learning. *Journal of Language Modelling*, August 2021. URL `https://jlm.ipipan.waw.pl/index.php/JLM/article/view/262`.

Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer Berlin / Heidelberg, 2004. doi: 10.1007/978-3-662-07003-1.

M. Lothaire. *Applied Combinatorics on Words*. Cambridge University Press, New York, 2005.

Connor Mayer and Travis Major. A challenge for tier-based strict locality from Uyghur backness harmony. In Annie Foret, Greg Kobele, and Sylvain Pogodalla, editors, *Formal Grammar 2018*, volume 10950 of *Lecture Notes in Computer Science*, pages 62–83. 2018. doi: 10.1007/978-3-662-57784-4_4.

John J. McCarthy. Feature geometry and dependency: A review. *Phonetica*, 45(2–4):84–108, 1988. doi: 10.1159/000261820.

Kevin McMullin. *Tier-Based Locality in Long-Distance Phonotactics: Learnability and Typology*. PhD thesis, University of British Columbia, 2016.

Robert McNaughton. Algebraic decision procedures for local testability. *Mathematical Systems Theory*, 8(1):60–76, March 1974. doi: 10.1007/bf01761708.

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

Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, EC-9(1):39–47, March 1960. doi: 10.1109/TEC.1960.5221603.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):268–311, June 1997.

Anil Nerode. Linear automaton transformations. In *Proceedings of the American Mathematical Society*, volume 9, pages 541–544. American Mathematical Society, August 1958. doi: 10.1090/s0002-9939-1958-0135681-9.

Daniel Nathan Osherson, Michael Stob, and Scott Weinstein. *Systems That Learn*. MIT Press, 1986.

Jean-Éric Pin. Syntactic semigroups. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*, pages 679–746. Springer-Verlag, Berlin, 1997. doi: 10.1007/978-3-642-59136-5_10.

Michael Oser Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, April 1959. doi: 10.1147/rd.32.0114.

Jonathan Rawski and Hossep Dolatian. Multi-input strictly local functions for tonal phonology. In *Proceedings of the Society for Computation in Linguistics*, volume 3, pages 245–260, New Orleans, Louisiana, 2020. URL `https://scholarworks.umass.edu/scil/vol3/iss1/25`.

James Rogers. A model-theoretic framework for theories of syntax. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 10–16, Santa Cruz, CA, 1996. Association for Computational Linguistics. doi: 10.3115/981863.981865.

James Rogers. Strict LT_2 : regular :: local : recognizable. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics: First International Conference, LACL '96 (Selected Papers)*, volume 1328 of *Lecture Notes in Computer Science*, pages 366–385, Berlin, 1997. Springer-Verlag. doi: 10.1007/BFb0052167.

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

James Rogers and Dakotah Lambert. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77, Toronto, Canada, July 2019a. Association for Computational Linguistics. doi: 10.18653/v1/W19-5706.

James Rogers and Dakotah Lambert. Extracting Subregular constraints from Regular stringsets. *Journal of Language Modelling*, 7(2):143–176, September 2019b. doi: 10.15398/jlm.v7i2.209.

James Rogers and Geoffrey K. Pullum. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20(3):329–342, June 2011. doi: 10.1007/s10849-011-9140-2.

James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language: Revised Selected Papers from the 10th and 11th Biennial Conference on the Mathematics of Language*, volume 6149 of *LNCS/LNAI*, pages 255–265. FoLLI/Springer, 2010. doi: 10.1007/978-3-642-14322-9_19.

James Rogers, Jeff Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. Cognitive and sub-regular complexity. In Glyn Morrill and Mark-Jan Nederhof, editors, *Formal Grammar 2012*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer-Verlag, 2012. doi: 10.1007/978-3-642-39998-5_6.

Ichirō Sakai. Syntax in universal translation. In *1961 International Conference on Machine Translation of Languages and Applied Language Analysis*, pages 593–608, London, 1962. Her Majesty's Stationery Office.

Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, April 1965. doi: 10.1016/s0019-9958(65)90108-7.

Marcel-Paul Schützenberger. Sur certaines opérations de fermeture dans les langages rationnels. *Symposia Mathematica*, 15:245–253, 1975.

Imre Simon. 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, 1975. doi: 10.1007/3-540-07407-4_23.

Noah A. Smith and Mark Johnson. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4):477–491, December 2007. doi: 10.1162/coli.2007.33.4.477.

Magnus Steinby. Rectangular algebras as tree recognizers. *Acta Cybernetica*, 22(2):499–515, January 2015. doi: 10.14232/actacyb.22.2.2015.15.

Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as powerful as one quantifier alternation: $\mathrm{FO}^2 = \Sigma_2 \cap \Pi_2$. In *STOC '98: Proceedings of the Thirtieth Annual ACM Symbosium on Theory of Computing*, pages 234–240, New York, NY, 1998. Association for Computing Machinery. doi: 10.1145/276698.276749.

Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and Systems Sciences*, 25:360–376, 1982. doi: 10.1016/0022-0000(82)90016-2.

Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, June 1968. doi: 10.1145/363347.363387.

Paola Valdivia, Paolo Buono, Catherine Plaisant, Nicole Dufournaud, and Jean-Daniel Fekete. Analyzing dynamic hypergraphs with parallel aggregated ordered hypergraph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(1):1–13, January 2021. doi: 10.1109/TVCG.2019.2933196.

Leslie Gabriel Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984. doi: 10.1145/1968.1972.

Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Logical Methods in Computer Science*, 5(3):1–23, August 2009. doi: 10.2168/LMCS-5(3:4)2009.

Charles Yang. Negative knowledge from positive evidence. *Language*, 91(4):938–953, 2015. doi: 10.1353/lan.2015.0054.

Daniel Haven Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, February 1967. doi: 10.1016/S0019-9958(67)80007-X.