Theoretical Computational Linguistics: Finite-state Automata

Jeffrey Heinz

January 22, 2023

Contents

Chapter 1

Introduction

1.1 Computational Linguistics: Course Overview

In this class, we will study:

1. Formal Language Theory
2. Automata Theory
3. Haskell
4. ...as they pertain to problems in linguistics:
   (a) *Well-formedness* of linguistic representations
   (b) *Transformations* from one representation to another

1.1.1 Linguistic Theory

Linguistic theory often distinguishes between *well- and ill-formed representations.*

Strings. In English, we can coin new words like *bling.* What about the following?

1. *gding*
2. *θwɪk*
3. *spɪf*

Trees. In English, we interpret the compound *deer-resistant* as an adjective, not a noun. What about the following?

1. *green-house*
2. *dry-clean*
3. *over-throw*

Linguistic theory is often also concerned with *transformations.*

Strings. In generative phonology, underlying representations of words are *transformed* to surface representations of words.

1. /kæt-z/ → [kæts]
2. /wɪʃ-z/ → [wɪʃɪz]

Trees. In derivational theories of generative syntax, the deep sentence structure is *transformed* into a surface structure.

1. Mary won the competition.
   (a) The competition was won by Mary.
   (b) What did Mary win?

1.1.2 Automata Theory

Automata are *abstract* machines that answer questions like these.

The Membership Problem

Given: A possibly infinite set of strings (or trees) X.

Input: A input string (or tree) x.

Problem: Does x belong to X?

The Transformation Problem

Given: A possible infinite function of strings to strings (or trees to trees) $f : X \to Y$.

Input: A input string (or tree) x.

Problem: What is $f(x)$?

There are many kinds of automata. Two common types of automata address these specific problems.

Recognizers Recognizers solve the membership problem.
Transducers Transducers solve the transformation problem.

Different kinds of automata instantiate different kinds of memory.

Finite-state Automata An automata is finite-state whenever the amount of memory necessary to solve a problem for input x is fixed and **independent** of the size of x.

Linear-bounded Automata An automata is linear-bounded whenever the amount of memory necessary to solve a problem for input x is **bounded by a linear function** of the size of x.

In this class we will study finite-state recognizers and transducers. There are many types of these as well, some are shown below.

- deterministic vs. non-deterministic
- 1way vs. 2way (for strings)
- bottom-up vs. top-down vs. walking (for trees)

The simplest type is the deterministic, 1way recognizer for strings. We will start there and then complicate them bit by bit:

1. add non-determinism
2. add output (transducers)
3. add 2way-ness
4. generalize strings to trees and repeat

What do automata mean for linguistic theory?

Fact 1: Finite-state automata over strings are sufficient for phonology and morphology (Johnson, 1972; Kaplan and Kay, 1994; Roark and Sproat, 2007; Dolatian and Heinz, 2020).

Fact 2: Finite-state automata over strings are *NOT* sufficient for syntax, but linear-bounded automata are (Chomsky, 1956; Huybregts, 1984; Shieber, 1985, among others).

Fact 3: Finite-state automata over trees *ARE* sufficient for syntax (Rogers, 1998; Kobele, 2011; Graf, 2011; Stabler, 2019).

Hypothesis: Linguistic phenomena can be modeled with special kinds of finite-state automata with even stricter memory requirements over the right representations (Heinz, 2018; Graf and De Santo, 2019; Graf, 2022).

Chapter 2

Formal Language Theory

The material in this chapter is covered in much greater detail in a number of textbooks including McNaughton and Papert (1971); Harrison (1978); Hopcroft *et al.* (1979); Davis and Weyuker (1983); Hopcroft *et al.* (2001) and Sipser (1997). Here we will state definitions and theorems, but we will not cover the proofs of the theorems.

We begin with the following question: If we choose to model natural languages with formal languages, what kind of formal languages are they? We have some idea what natural languages are. After all, you are reading this! A satisfactory answer to answer this question however also requires being clear about what a formal language is.

2.1 Formal Languages

A formal language is a set of strings. Strings are sequences of symbols of finite length. The symbol Σ commonly denotes a finite set of symbols. There is a unique string of length zero, which is the empty string. This is commonly denoted with λ or ϵ.

A key operation on strings is *concatenation*. The concatenation of string x with string y is written xy. Concatenation is associative: for all strings x, y, z, it holds that $(xy)z = x(yz)$. The empty string is an identity element for concatenation: for all strings x, $x\lambda = \lambda x = x$. If we concatenate a string x with itself n times we write x^n. For example, $(ab)^3 = ababab$.

We can also concatenate two formal languages X and Y.

$$XY = \{xy : x \in X, y \in Y\}$$

Language concatenation is also associative. The empty string language $\{\lambda\}$ is an identity element for language concatenation: for all languages L, $L\{\lambda\} = \{\lambda\}L = L$. Also, the empty set $\varnothing$ is a zero element for language concatenation: for all languages L, $L\varnothing = \varnothing L = \varnothing$.

If we concatenate a language X with itself n times we write X^n. For example, $XX = X^2$. Finally for any language X, we define X^* as follows.

$$X^* = \{\lambda\} \cup X \cup X^2 \cup X^3 \ldots = \bigcup_{n\geq 0} X^n$$

where X^0 is defined as $\{\lambda\}$. The asterisk (*) is called the Kleene star after Kleene (1956) who introduced it.

It follows that the set of all strings of finite length can be denoted Σ^*. Consequently formal languages can be thought of as subsets of Σ^*. How can we talk about such subsets?

One way is to use set notation and set construction. Example 1 present some examples of formal languages defined in these ways.

Example 1. In this example, assume $\Sigma = \{a, b, c\}$.

1. $\{\lambda, a\}$.
2. $\{\lambda, a, aa\}$.
3. $\{a^n \in \Sigma^* : n \leq 10\}$.
4. $\{a^n \in \Sigma^* : n \geq 0\}$.
5. $\{w \in \Sigma^*\}$.
6. $\{w \in \Sigma^* : w \textit{ contains the string } aa\}$.
7. $\{w \in \Sigma^* : w \textit{ does not contain the string } aa\}$.
8. $\{w \in \Sigma^* : w \textit{ contains a } b \textit{ somewhere after an } a\}$.
9. $\{w \in \Sigma^* : w \textit{ does not contain a } b \textit{ somewhere after an } a\}$.
10. $\{w \in \Sigma^* : w \textit{ contains either the string } aa \textit{ or the string } bb\}$.
11. $\{w \in \Sigma^* : w \textit{ contains both the string } aa \textit{ and the string } bb\}$.
12. $\{w \in \Sigma^* : w \textit{ does not contain the string } bb \textit{ on the } \{b,c\} \textit{ tier}\}$.
13. $\{w \in \Sigma^* : w \textit{ contains an even number of } as\}$.
14. $\{a^n b^n \in \Sigma^* : n \geq 1\}$.
15. $\{a^n b^m \in \Sigma^* : m > n\}$.
16. $\{a^n b^n c^n \in \Sigma^* : n \geq 1\}$.
17. $\{a^n b^m c^\ell \in \Sigma^* : \ell > m > n\}$.
18. $\{w \in \Sigma^* : \textit{the number of } bs \textit{ is the same as the number of } cs \textit{ in } w\}$.
19. $\{w \in \Sigma^* : \textit{the number of } as, bs, \textit{ and } cs \textit{ is the same in } w\}$.
20. $\{a^n \in \Sigma^* : n \textit{ is a prime number}\}$.

2.2 Grammars

There are two important aspects to defining grammar formalisms. They are distinct, but related, aspects.

1. The grammar itself. This is an object and in order to be well-formed it has to follow certain rules and/or conditions.

2. How the grammar is associated with a language. A separate set of rules/conditions explains how to *interpret* the grammar. This aspect explains how the grammar *generates/recognizes/accepts* a language.

In other words, by itself, a grammar is more or less useless. But combined with a way to interpret it—a way to associate it with a formal language—it becomes a very powerful form of expression.

2.3 Expression Grammars

As a first example, consider regular expressions. These consist of both a syntax (which define well-formed regular expressions) and a semantics (which associate them unambiguously with formal languages). They are defined inductively.

Syntax		**Semantics**		
REs include				
• each $\sigma \in \Sigma$	*(singleton letter set)*	$[\![\sigma]\!]$	$=$	$\{\sigma\}$
• ϵ	*(empty string set)*	$[\![\epsilon]\!]$	$=$	$\{\epsilon\}$
• $\varnothing$	*(empty set)*	$[\![\varnothing]\!]$	$=$	$\{\}$
If R, S are REs then so are:				
• (R∘S)	*(concatenation)*	$[\![(R \cdot S)]\!]$	$=$	$[\![R]\!] \circ [\![S]\!]$
• (R+S)	*(union)*	$[\![(R + S)]\!]$	$=$	$[\![R]\!] \cup [\![S]\!]$
• (R^*)	*(Kleene star)*	$[\![(R^*)]\!]$	$=$	$[\![R]\!]^*$

We say a language is *regular* if there is a regular expression denoting it. The class of regular languages is denoted $[\![RE]\!]$.

Exercise 1. Write regular expressions for as many of the languages in Example 1 as you can.

2.3.1 Cat-Union Expressions

Syntax		**Semantics**		
CUEs include				
• each $\sigma \in \Sigma$	*(singleton letter set)*	$[\![\sigma]\!]$	$=$	$\{\sigma\}$
• ϵ	*(empty string set)*	$[\![\epsilon]\!]$	$=$	$\{\epsilon\}$
• $\varnothing$	*(empty set)*	$[\![\varnothing]\!]$	$=$	$\{\}$
If R, S are CUEs then so are:				
• (R∘S)	*(concatenation)*	$[\![(R \cdot S)]\!]$	$=$	$[\![R]\!] \circ [\![S]\!]$
• (R+S)	*(union)*	$[\![(R + S)]\!]$	$=$	$[\![R]\!] \cup [\![S]\!]$

So CUEs are a fragment of REs that exclude Kleene star.

Exercise 2. Write CUEs for as many of the languages in Example 1 as you can. What kinds of formal languages do cat-union expressions describe?

Theorem 1. $[\![CUE]\!] = \{L \subseteq \Sigma^* : |L| \text{ is finite}\} \subsetneq [\![RE]\!]$

2.3.2 Generalized Regular Expressions

Syntax		**Semantics**		
GREs include				
• each $\sigma \in \Sigma$	*(singleton letter set)*	$[\![\sigma]\!]$	$=$	$\{\sigma\}$
• ϵ	*(empty string set)*	$[\![\epsilon]\!]$	$=$	$\{\epsilon\}$
• $\varnothing$	*(empty set)*	$[\![\varnothing]\!]$	$=$	$\{\}$
If R, S are GREs then so are:				
• (R∘S)	*(concatenation)*	$[\![(R \cdot S)]\!]$	$=$	$[\![R]\!] \circ [\![S]\!]$
• (R+S)	*(union)*	$[\![(R + S)]\!]$	$=$	$[\![R]\!] \cup [\![S]\!]$
• (R^*)	*(Kleene star)*	$[\![(R^*)]\!]$	$=$	$[\![R]\!]^*$
• (R&S)	*(intersection)*	$[\![(R\&S)]\!]$	$=$	$[\![R]\!] \cap [\![S]\!]$
• ($\overline{\mathrm{R}}$)	*(complement)*	$[\![\overline{\mathrm{R}}]\!]$	$=$	$\Sigma^* - [\![R]\!]$

Theorem 2. $[\![RE]\!] = [\![GRE]\!]$

Exercise 3. Write GREs for as many of the languages in Example 1 as you can.

2.3.3 Star Free Expressions

Syntax		**Semantics**		
SFEs include				
• each $\sigma \in \Sigma$	*(singleton letter set)*	$[\![\sigma]\!]$	$=$	$\{\sigma\}$
• ϵ	*(empty string set)*	$[\![\epsilon]\!]$	$=$	$\{\epsilon\}$
• $\varnothing$	*(empty set)*	$[\![\varnothing]\!]$	$=$	$\{\}$
If R, S are SFEs then so are:				
• (R∘S)	*(concatenation)*	$[\![(R \cdot S)]\!]$	$=$	$[\![R]\!] \circ [\![S]\!]$
• (R+S)	*(union)*	$[\![(R + S)]\!]$	$=$	$[\![R]\!] \cup [\![S]\!]$
• (R&S)	*(intersection)*	$[\![(R \& S)]\!]$	$=$	$[\![R]\!] \cap [\![S]\!]$
• ($\overline{\text{R}}$)	*(complement)*	$[\![\overline{\text{R}}]\!]$	$=$	$\Sigma^* - [\![R]\!]$

So SFEs are a fragment of GREs that exclude Kleene star.

Exercise 4. Write SFEs for as many of the languages in Example 1 as you can.

Theorem 3. $[\![CUE]\!] \subsetneq [\![SFE]\!] \subsetneq [\![RE]\!] = [\![GRE]\!]$

For more information on the theorems in this section, see McNaughton and Papert (1971).

2.3.4 Piecewise Local Expressions

Dakotah Lambert developed PLEs over the past ten years. His 2022 dissertation provides a written treatment. I present a large fragment of them here (some more details are in the thesis). Part of the motivation for PLEs is to develop linguistically motivated expression-builders.

As an example, Lambert introduces a tier operator which takes two arguments: a set of symbols T (the tier elements) and a language L. Non-tier elements are freely insertable and deleteable (they have no effect on whether a string belongs to the language or not). Removing the non-tier symbols from a word yields a string of symbols on the tier. Given a language L, let us call the language obtained from removing the non-tier symbols from all of its words, the tier-projection of L. Then Lambert's tier operator produces the largest language in Σ^* such that its tier-projection equals the tier-projection of L. Lambert's operator is thus the *maximal, inverse* tier-projection.

Formally, for all $\sigma \in \Sigma$ and all $T \subseteq \Sigma$, let $I_T(\sigma)$ denote the string σ iff $\sigma \in T$ and λ otherwise. Then, for all $w = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*$, we let let $[T]w$ be the language $S^* I_T(\sigma_1)S^* I_T(\sigma_2)S^* \ldots S^* I_T(\sigma_n)S^*$ where $S = \Sigma - T$. Finally, for any language L, we let $[T]L = \bigcup_w \in L[T]w$.

Syntax		**Semantics**		
For all $\sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*$ PLEs include				
• $\langle\sigma_1\sigma_2\ldots\sigma_n\rangle$	*(unanchored substring)*	$[\![\langle\sigma_1\sigma_2\ldots\sigma_n\rangle]\!]$	=	$\Sigma^*\sigma_1\sigma_2\ldots\sigma_n\Sigma^*$
• $\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle$	*(unanchored subsequence)*	$[\![\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle]\!]$	=	$\Sigma^*\sigma_1\Sigma^*\sigma_2\Sigma^*\ldots\Sigma^*\sigma_n\Sigma^*$
• $\rtimes\langle\sigma_1\sigma_2\ldots\sigma_n\rangle$	*(left-anchored substring)*	$[\![\rtimes\langle\sigma_1\sigma_2\ldots\sigma_n\rangle]\!]$	=	$\sigma_1\sigma_2\ldots\sigma_n\Sigma^*$
• $\rtimes\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle$	*(left-anchored subsequence)*	$[\![\rtimes\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle]\!]$	=	$\sigma_1\Sigma^*\sigma_2\Sigma^*\ldots\Sigma^*\sigma_n\Sigma^*$
• $\ltimes\langle\sigma_1\sigma_2\ldots\sigma_n\rangle$	*(right-anchored substring)*	$[\![\ltimes\langle\sigma_1\sigma_2\ldots\sigma_n\rangle]\!]$	=	$\Sigma^*\sigma_1\sigma_2\ldots\sigma_n$
• $\ltimes\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle$	*(right-anchored subsequence)*	$[\![\ltimes\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle]\!]$	=	$\Sigma^*\sigma_1\Sigma^*\sigma_2\Sigma^*\ldots\Sigma^*\sigma_n$
• $\rtimes\ltimes\langle\sigma_1\sigma_2\ldots\sigma_n\rangle$	*(anchored substring)*	$[\![\rtimes\ltimes\langle\sigma_1\sigma_2\ldots\sigma_n\rangle]\!]$	=	$\{\sigma_1\sigma_2\ldots\sigma_n\}$
• $\rtimes\ltimes\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle$	*(anchored subsequence)*	$[\![\rtimes\ltimes\langle\sigma_1,\sigma_2,\ldots,\sigma_n\rangle]\!]$	=	$\sigma_1\Sigma^*\sigma_2\Sigma^*\ldots\Sigma^*\sigma_n$
If $R_1, R_2, \ldots R_n$ are PLEs then so are:				
• $\neg R_1$	*(complement)*	$[\![\neg R_1]\!]$	=	$\Sigma^* - [\![R_1]\!]$
• $*R_1$	*(Kleene star)*	$[\![*R_1]\!]$	=	$[\![R_1]\!]^*$
• $[\sigma_1,\sigma_2,\ldots\sigma_n]R_1$	*(tier max-inv-projection)*	$[\![[\sigma_1,\sigma_2,\ldots\sigma_n]R_1]\!]$	=	$[\sigma_1,\sigma_2,\ldots\sigma_n]\,[\![R_1]\!]$
• $\wedge\{R_1,R_2,\ldots R_n\}$	*(intersection)*	$[\![\wedge\{R_1,R_2,\ldots R_n\}]\!]$	=	$\bigcap_{1\leq i\leq n}[\![R_i]\!]$
• $\vee\{R_1,R_2,\ldots R_n\}$	*(union)*	$[\![\vee\{R_1,R_2,\ldots R_n\}]\!]$	=	$\bigcup_{1\leq i\leq n}[\![R_i]\!]$
• $\circ\{R_1,R_2,\ldots R_n\}$	*(concatenation)*	$[\![\circ\{R_1,R_2,\ldots R_n\}]\!]$	=	$[\![R_1]\!]\circ[\![R_2]\!]\circ\ldots\circ[\![R_n]\!]$

Theorem 4. $[\![CUE]\!] \subsetneq [\![SFE]\!] \subsetneq [\![RE]\!] = [\![GRE]\!] = [\![PLE]\!]$

Exercise 5. Write PLEs for as many of the languages in Example 1 as you can.

2.4 Rewrite Grammars

There are many ways to define grammars which describe formal languages. Another influential approach has been rewrite grammars (Hopcroft *et al.*, 1979).

Definition 1. *A* rewrite grammar[1] *is a tuple $\langle T, N, S, \mathcal{R}\rangle$ where*

- $\mathcal{T}$ *is a nonempty finite alphabet of symbols. These symbols are also called the* terminal *symbols, and we usually write them with lowercase letters like $a, b, c, \ldots$*

[1]For a slightly different definition and some more description of rewrite grammars, see Partee *et al.* (1993, chap. 16).

- $\mathcal{N}$ *is a nonempty finite set of* non-terminal *symbols, which are distinct from elements of* $\mathcal{T}$*. These symbols are also called* category *symbols, and we usually write them with uppercase letters like* $A, B, C, \ldots$
- S *is the* start *category, which is an element of* $\mathcal{N}$*.*
- *A finite set of* production rules $\mathcal{R}$*. A production rule has the form*

$$\alpha \rightarrow \beta$$

where α, β *belong to* $(\mathcal{T} \cup \mathcal{N})^*$*. In other words,* α *and* β *are strings of non-terminal and terminal symbols. While* β *may be the empty string we require that* α *include at least one symbol.*

Rewrite grammars are also called *phrase structure grammars*.

Example 2. Consider the following grammar G_1:

- $\mathcal{T} = \{$***john***, ***laughed***, ***and***$\}$;
- $\mathcal{N} = \{$S, VP1, VP2$\}$; and
-

$$\mathcal{R} = \left\{ \begin{array}{l} \text{S} \rightarrow \textbf{\textit{john}} \text{ VP1} \\ \text{VP1} \rightarrow \textbf{\textit{laughed}} \\ \text{VP1} \rightarrow \textbf{\textit{laughed}} \text{ VP2} \\ \text{VP2} \rightarrow \textbf{\textit{and laughed}} \\ \text{VP2} \rightarrow \textbf{\textit{and laughed}} \text{ VP2} \end{array} \right\}$$

Example 3. Consider the following grammar G_2:

- $\mathcal{T} = \{$a, b$\}$;
- $\mathcal{N} = \{$S, A, B$\}$; and
-

$$\mathcal{R} = \left\{ \begin{array}{l} \text{S} \rightarrow \text{ABS} \\ \text{S} \rightarrow \lambda \\ \text{AB} \rightarrow \text{BA} \\ \text{BA} \rightarrow \text{AB} \\ \text{A} \rightarrow \text{a} \\ \text{B} \rightarrow \text{b} \end{array} \right\}$$

Example 4. Consider the following grammar G_3:

- $\mathcal{T} = \{$a, b$\}$;
- $\mathcal{N} = \{$S$\}$; and

- $$\mathcal{R} = \left\{ \begin{array}{l} \mathrm{S} \rightarrow \mathrm{ba} \\ \mathrm{S} \rightarrow \mathrm{baba} \\ \mathrm{S} \rightarrow \mathrm{bab} \end{array} \right\}$$

The language of a rewrite grammar is defined recursively below.

Definition 2. *The (partial)* derivations *of a rewrite grammar $G = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ is written $D(G)$ and is defined recursively as follows.*

1. The base case: *S belongs to $D(G)$.*
2. The recursive case: *For all $\alpha \rightarrow \beta \in \mathcal{R}$ and for all $\gamma_1, \gamma_2 \in (\mathcal{T} \cup \mathcal{N})^*$, if $\gamma_1 \alpha \gamma_2 \in D(G)$ then $\gamma_1 \beta \gamma_2 \in D(G)$.*
3. *Nothing else is in $D(G)$.*

Then the language of the grammar $L(G)$ is defined as

$$L(G) = \{ w \in \mathcal{T}^* : w \in D(G) \}.$$

Exercise 6. How does G_1 generate ***John laughed and laughed and laughed***?

Exercise 7. What language does G_2 generate?

Exercise 8. What language does G_3 generate?

2.5 The Chomsky Hierarchy

"By putting increasingly stringent restrictions on the allowed forms of rules we can establish a series of grammars of decreasing generative power. Many such series are imaginable, but the one which has received the most attention is due to Chomsky and has come to be known as the Chomsky Hierarchy." (Partee *et al.*, 1993, p. 451)

Recall that rules are of the form $\alpha \rightarrow \beta$ with $\alpha, \beta \in (\mathcal{T} \cup \mathcal{N})^*$, with the further restriction that α was not the empty string.

Type 0 There is no further restriction on α or β.

Type 1 Each rule is of the form $\alpha \rightarrow \beta$ where α contains at least one symbol $A \in \mathcal{N}$ and β is not the empty string.

Type 2 Each rule is of the form $A \rightarrow \beta$ where $A \in \mathcal{N}$ and $\beta \in (\mathcal{T} \cup \mathcal{N})^*$.

Type 3 Each rule is of the form $A \rightarrow aB$ or $A \rightarrow a$ where $A, B \in \mathcal{N}$ and $a \in \mathcal{T}$.

There is one exception to the above restrictions for Types 1, 2 and 3. For these types, the production $S \rightarrow \lambda$ is allowed. If this production is included in a grammar then the formal language it describes will include the empty string. Otherwise, it will not.

To this we will add an additional type which we will call finite:

finite Each rule of is of the form $S \rightarrow w$ where $w \in \mathcal{T}^*$.

Each of these types goes by other names.

Type 0	recursively enumerable, computably enumerable
Type 1	context-sensitive
Type 2	context-free
Type 3	regular, right-linear[2]
finite	finite

Table 2.1: Names for classes of formal languages.

These names refer to both the *grammars* and the *languages*. These are different kinds of objects, so it is important to know which one is being referred to in any given context.

Theorem 5 (Chomsky Hierarchy). *1.* $[\![type-3]\!] \subseteq [\![type-2]\!]$ *(Scott and Rabin, 1959).*

2. $[\![type-2]\!] \subseteq [\![type-1]\!]$ *(Bar-Hillel* et al.*, 1961).*

3. $[\![type-1]\!] \subseteq [\![type-0]\!]$ [3]

For details, see, for instance, Davis and Weyuker (1983).

Exercise 9. Write rewrite grammars for as many of the languages in Example 1 as you can. Are they type 1, 2, 3 or finite grammars?

If we choose to model natural languages with formal languages, what kind of formal languages are they?

[2]Technically, right-linear grammars are defined as those languages where each rule is of the form $A \rightarrow aB$ or $A \rightarrow a$ where $A, B \in \mathcal{N}$ and $a \in \mathcal{T}$. Consequently is not possible for a right linear grammar to define a language which includes the empty string.

[3]This is a diagnolization argument of the kind originally due to Cantor. Rogers (1967) is a good source for this kind of thing.

Bibliography

Bar-Hillel, Y., M. Perles, and E. Shamir. 1961. On formal properties of simple phrase-structure grammars. *Zeitschrift fur Phonetik, Sprachwissenschaft, und Kommunikationsforschung* 14:143–177.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory* 113–124. IT-2.

Davis, Martin D., and Elaine J. Weyuker. 1983. *Computability, Complexity and Languages*. Academic Press.

Dolatian, Hossep, and Jeffrey Heinz. 2020. Computing and classifying reduplication with 2-way finite-state transducers. *Journal of Language Modelling* 8:179–250.

Graf, Thomas. 2011. Closure properties of Minimalist derivation tree languages. In *LACL 2011*, edited by Sylvain Pogodalla and Jean-Philippe Prost, vol. 6736 of *Lecture Notes in Artificial Intelligence*, 96–111. Heidelberg: Springer.

Graf, Thomas. 2022. Subregular linguistics: bridging theoretical linguistics and formal grammar. *Theoretical Linguistics* 48:145–184.

Graf, Thomas, and Aniello De Santo. 2019. Sensing tree automata as a model of syntactic dependencies. In *Proceedings of the 16th Meeting on the Mathematics of Language*, 12–26. Toronto, Canada: Association for Computational Linguistics.
URL `https://aclanthology.org/W19-5702`

Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley Publishing Company.

Heinz, Jeffrey. 2018. The computational nature of phonological generalizations. In *Phonological Typology*, edited by Larry Hyman and Frans Plank, Phonetics and Phonology, chap. 5, 126–195. De Gruyter Mouton.

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

Hopcroft, John, Rajeev Motwani, and Jeffrey Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*. Boston, MA: Addison-Wesley.

Huybregts, Riny. 1984. The weak inadequacy of context-free phrase structure grammars. In *Van periferie naar kern*, edited by Ger de Haan, Mieke Trommelen, and Wim Zonneveld, 81–99. Dordrecht, The Netherlands: Foris.

Johnson, C. Douglas. 1972. *Formal Aspects of Phonological Description*. The Hague: Mouton.

Kaplan, Ronald, and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Kleene, S.C. 1956. Representation of events in nerve nets. In *Automata Studies*, edited by C.E. Shannon and J. McCarthy, 3–40. Princeton University. Press.

Kobele, Gregory M. 2011. Minimalist tree languages are closed under intersection with recognizable tree languages. In *LACL 2011*, edited by Sylvain Pogodalla and Jean-Philippe Prost, vol. 6736 of *Lecture Notes in Artificial Intelligence*, 129–144. Berlin: Springer.

Lambert, Dakotah. 2022. Unifying classification schemes for languages and processes with attention to locality and relativizations thereof. Doctoral dissertation, Stony Brook University.
URL `https://vvulpes0.github.io/PDF/dissertation.pdf/`

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

Partee, Barbara, Alice ter Meulen, and Robert Wall. 1993. *Mathematical Methods in Linguistics*. Dordrect, Boston, London: Kluwer Academic Publishers.

Roark, Brian, and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford: Oxford University Press.

Rogers, Hartley. 1967. *Theory of Recursive Functions and Effective Computability*. McGraw Hill Book Company.

Rogers, James. 1998. *A Descriptive Approach to Language-Theoretic Complexity*. Stanford, CA: CSLI Publications.

Scott, Dana, and Michael Rabin. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development* 5:114–125.

Shieber, Stuart. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8:333–343.

Sipser, Michael. 1997. *Introduction to the Theory of Computation*. PWS Publishing Company.

Stabler, Edward P. 2019. Three mathematical foundations for syntax. *Annual Review of Linguistics* 5:243–260.