Introduction to Formal Language Theory

Jeffrey Heinz

February 3, 2021

The material in this handout is covered in much greater detail in a number of textbooks including Harrison (1978); 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.

1 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 a 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 language—it becomes a very powerful form of expression.

2 Rewrite Grammars

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* $a, b, c, \ldots$

- $\mathcal{N}$ *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* $A, B, C, \ldots$

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

- *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*

$$\alpha \to \beta$$

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 1 *Consider the following grammar G_1:*

- $\mathcal{T} = \{\textbf{\textit{john}}, \textbf{\textit{laughed}}, \textbf{\textit{and}}\}$*;*
- $\mathcal{N} = \{S,\ VP1,\ VP2\}$*; and*
- $$\mathcal{R} = \left\{ \begin{array}{l} S \to \textbf{\textit{john}}\ VP \\ VP1 \to \textbf{\textit{laughed}}\ VP2 \\ VP2 \to \textbf{\textit{and laughed}}\ VP2 \\ VP2 \to \lambda \end{array} \right\}$$

Example 2 *Consider the following grammar G_2:*

- $\mathcal{T} = \{a,\ b\}$*;*
- $\mathcal{N} = \{S,\ A,\ B\}$*; and*
- $$\mathcal{R} = \left\{ \begin{array}{l} S \to ABS \\ S \to \lambda \\ AB \to BA \\ BA \to AB \\ A \to a \\ B \to b \end{array} \right\}$$

Example 3 *Consider the following grammar G_3:*

- $\mathcal{T} = \{a,\ b\}$*;*
- $\mathcal{N} = \{S\}$*; and*
- $$\mathcal{R} = \left\{ \begin{array}{l} S \to ba \\ S \to baba \\ S \to bab \end{array} \right\}$$

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* $\alpha \to \beta \in \mathcal{R}$ *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*

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

Exercise 1 *How does* G_1 *generate* ***John laughed and laughed and laughed****?*

Exercise 2 *What language does* G_2 *generate?*

Exercise 3 *What language does* G_3 *generate?*

3 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 $\alpha \to \beta$ 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 $\alpha \to \beta$ where β is not the empty string.

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

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

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

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

Each of these types goes by other names.

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.

[2]Technically, right-linear grammars are defined as those languages where each rule is of the form $A \to aB$ or $A \to a$ 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.

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

Table 1: Names for classes of formal languages.

Theorem 1 (Chomsky Hierarchy) *1. The languages generated by type-3 grammars are a proper subset of the languages generated by type-2 grammars (Scott and Rabin, 1959).*

2. The languages generated by type-2 grammars are a proper subset of the languages generated by type-1 grammars (leaving aside the empty string) (Bar-Hillel et al.*, 1961).*

3. The languages generated by type-1 grammars are a proper subset of the languages generated by type-0 grammars.[3]

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

Exercise 4 *1. Write a type 3 grammar which generates the language*

$$L = \{cv, cvcv, cvcvcv, \ldots\}.$$

Let's call this the "CV" language: the language where every word begins with a c*, ends with a* v*, and alternates* cs *and* vs*. This is an infinite language.*

2. Write a type 2 grammar which generates the language L=$\{\lambda$*, ab, aabb, aaabbb,* $\ldots\}$*. This language is often called* $a^n b^n$ *since it begins with zero or more* as *which are followed by a matching number of* bs*. It's also an infinite language.*

3. Try to write a type 3 grammar for the language $a^n b^n$*. What problems do you run into?*

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

References

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.

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

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

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

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

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

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

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

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