— LECTURE NOTES —

Formal Languages

1 The classical perspective

1 **Vocabulary** Let Σ be an arbitrary *finite* set. We will refer to Σ as our *vocabulary.*

2 **Strings** Let $[1,n]$ denote the sequence of integers $1, 2, \dots, n$. If $n < 1$, the sequence is empty.

Define a *string* w to be a function from $[1,n]$ to Σ, for some finite n. We usually write w_1 and w_2 in place of $w(1)$ and $w(2)$.

The string over $[1,0]$ is called *the empty string*, denoted ϵ.

We will write $|w|$ for the *length* of w ($|\epsilon| = 0$).

We generally do not distinguish between elements of Σ (words) and strings (sentences) of length 1.

3 **Concatenation** Define the *concatenation* of u and v, uv as follows. For all $i \in [1, |u| + |v|]$,

$$uv(i) = \begin{cases} u(i) & \text{if } i \leq |u| \\ v(i - |u|) & \text{otherwise} \end{cases}$$

Fact: Concatenation is *associative* ($(uv)w = u(vw)$).
Fact: The empty string ϵ is the identity element for concatenation ($\epsilon u = u$ and $u\epsilon = u$).

4 **Languages** Let Σ^* denote the set of all strings over Σ. Let Σ^+ denote the set of all *nonempty* strings over Σ, i.e., $\Sigma^* - \{\epsilon\}$.

A Σ-*language* L is a subset of Σ^*. Normally Σ is understood, and we just refer to L as a language.

5 **Operations on Languages** Given two languages L_1 and L_2, we can define a new language L_1L_2 as follows:

$$L_1L_2 := \{w \mid (\exists u \in L_1)(\exists v \in L_2)[w = uv]\}$$

Question: Under what conditions will it be the case that $\epsilon \in L_1L_2$?

Languages are sets, so the standard set theoretic operations all apply to languages. Given U and V we can make new languages by constructing $U \cap V$, $U \cup V$ and $U - V$, for example. We define the *complement* of L, L', as $\Sigma^* - L$.

We will often write the set theoretic union of L_1 and L_2 as $L_1 \mid L_2$, and refer to the operation as *alternation.*

Finally, we can form the "Kleene closure" of a language L, written L^*, by the following closure definition.

$$L^0 := \{\epsilon\} \tag{1}$$

$$L^{i+1} := L^i L \tag{2}$$

$$L^* := \bigcup_{i=0}^{\infty} L^i \tag{3}$$

The unary postfix operator * is often called the "Kleene star" operator. We also refer to this operation as *iteration*.

6 Defining Infinite Languages: The Regular Case Let $\mathcal{REG}$ denote the smallest set such that the following conditions hold.

1. $\mathcal{REG}$ contains all the finite languages.
2. $\mathcal{REG}$ is closed under the operations of concatenation, alternation and iteration.

A language which is in $\mathcal{REG}$ is called a *regular language*.
Fact: All of our operations except concatenation are just ordinary set theoretic operations. Therefore the notion of *regularity* is not just confined to sets of strings. Whenever we have a set of objects on which we can define an operator with the properties of concatenation, we can define a notion of a regular set of such objects, by taking the finite sets and closing them under the relevant operations.

7 Regular Expressions Given $a \in \Sigma$ let $\mathbf{a}$ denote the language containing only the string a. Then we can define the *regular expressions* as all of those terms that we can construct using for our constants (1) $\mathbf{a}$ for all $a \in \Sigma$ and (2) the special elements ϵ and $\emptyset$, and using $\circ$, $\cup$ and $*$ as our operators. (Alternatively, we may use simple juxtaposition for concatenation and the vertical bar $|$ for alternation.)
Fact: Each regular expression denotes a language in $\mathcal{REG}$, and any language in $\mathcal{REG}$ can be denoted by some regular expression (infinitely many, in fact, since $E|E$ denotes the same language as E, for any regular expression E).
Question: Given languages A and B, does $(A|B)^* = (A^*)|(B^*)$? Does $(AB)^* = A^*B^*$?

8 String Rewriting Assume now that we are given, in addition to Σ, a new set N, also finite. We will think of N as providing us with a set of (names for) *grammatical categories*. We want each element A of N to denote a set of strings in Σ^*, intuitively the set of strings of category A. These categories receive a recursive definition as follows.

The regular expression $(\Sigma \cup N)^* N (\Sigma \cup N)^*$ denotes the set of all strings over Σ and N that contain at least one element of N somewhere in them.

Let P denote a relation

$$P \subseteq ((\Sigma \cup N)^* N (\Sigma \cup N)^*) \times (\Sigma \cup N)^*$$

For strings α and β we usually write $(\alpha \longrightarrow \beta) \in P$ rather than $P(\alpha, \beta)$.

An element $(\alpha \longrightarrow \beta)$ of P is called a *production* of P. A production $(\alpha \longrightarrow \beta)$ *applies* to a string $\gamma_1 \alpha \gamma_2$ to yield a new string $\gamma_1 \beta \gamma_2$. In this case β is *substituted for* α in $\gamma_1 \alpha \gamma_2$.

Note that this is a funny sort of substitution: if a string γ contains multiple instances of α, and P contains several productions with α on the left hand side, then each separate instance of α in γ may be substituted for by a different string. This is unlike substitution for a variable in an expression: substituting 2 for x in $x * x$ should yield $2 * 2$, not $2 * x$ or $x * 2$.

As a notational convention, we will use u, v and w to refer to Σ-strings, and α, β and γ to refer to $(\Sigma \cup \mathrm{N})$-strings. Lowercase letters a, b and c will refer to elements of Σ, uppercase letters A, B, C, etc. will refer to elements of N.

9 Grammars A grammar is a 4-tuple $\langle \mathrm{N}, \Sigma, P, S \rangle$ where N, Σ and P are as above and S is an element of N called the *start symbol* of the grammar (or the *sentence category* of the grammar).

Let a *sentential form* generated by a grammar $G = \langle \mathrm{N}, \Sigma, P, S \rangle$ be defined as follows.

1. The string S is a sentential form.
2. If $\gamma_1 \alpha \gamma_2$ is a sentential form (generated by G) and $\alpha \longrightarrow \beta$ is a production in P, then $\gamma_1 \beta \gamma_2$ is a sentential form generated by G.
3. Nothing else is a sentential form generated by G.

The special case of a sentential form w containing no symbols from N is called a *sentence* generated by G. The *language generated by* G, denoted $L(G)$, is just the set of sentences generated by G.

The process of generating a sentence is called a *derivation*. We write $\gamma_1 \alpha \gamma_2 \Longrightarrow \gamma_1 \beta \gamma_2$ to represent a derivation step (sometimes with some decorations on the arrow to indicate what grammar or what production was used. '$\Longrightarrow$' is thus a relation over sentential forms, and we can form its transitive closure $\stackrel{+}{\Longrightarrow}$ and its reflexive transitive closure $\stackrel{*}{\Longrightarrow}$ in the usual ways.

10 Types of Grammars Given a grammar $G = \langle \mathrm{N}, \Sigma, P, S \rangle$, G is said to be

Right-linear if every production in P is of the form $A \longrightarrow wB$ or $A \longrightarrow w$.

Context-free if every production in P is of the form $A \longrightarrow \beta$.

Context-sensitive if every production in P is of the form $\alpha \longrightarrow \beta$, and $|\alpha| \leq |\beta|$. (In particular β cannot be ϵ. Therefore no context sensitive language contains ϵ.)

Unrestricted Otherwise.

If we stipulate that the "real" CSLs are obtained by closing those above under union with $\{\epsilon\}$, then these language families form a hierarchy in which each contains the ones above it. This hierarchy is called the *Chomsky Hierarchy.*

11 A language L is in $\mathcal{REG}$ if and only if it has a right linear grammar.

12 Phrase Structure Given a context-free grammar $G = \langle \mathrm{N}, \Sigma, P, S \rangle$, and assume that we have somehow indexed the productions so that we can refer to them, by name or number or some other scheme. Assume our indices are drawn from some set I. Then we can represent any derivation $\alpha \overset{*}{\Longrightarrow} \beta$ as a string in I^*.

Note, however, that we cannot quite recover the derivation from such a string. Given a sentential form $\gamma_1 A \gamma_2 A \gamma_3$, and a production $P_i = A \longrightarrow \beta$, we have both $\gamma_1 A \gamma_2 A \gamma_3 \Longrightarrow \gamma_1 \beta \gamma_2 A \gamma_3$ and $\gamma_1 A \gamma_2 A \gamma_3 \Longrightarrow \gamma_1 A \gamma_2 \beta \gamma_3$. So a string $viw \in I^*$ is ambiguous in its interpretation.

Let a *leftmost derivation* be a derivation in which we always expand the leftmost substring that matches a production's left hand side. Define analogously a *rightmost* derivation. This resolves our ambiguity.

But it leaves us with a notion of derivation which is *too* precise. Given productions $P_i = A \longrightarrow \alpha$ and $P_j = B \longrightarrow \beta$, and a sentential form $\gamma_1 A \gamma_2 B \gamma_3$, the derivations ij and ji will both leave us with the sentential form $\gamma_1 \alpha \gamma_2 \beta \gamma_3$. In this case, it really doesn't matter which order we apply things in.

Assume now that we decorate our productions with trees in the way hinted at below.

$$A \longrightarrow B\ C \qquad \rightsquigarrow \qquad \begin{array}{c} A \\ / \quad \backslash \\ t_B \qquad t_C \end{array} : A \longrightarrow t_B : B \quad t_C : C$$

Then the derivation $t : S \overset{*}{\Longrightarrow} w$ will leave us with a tree in t that represents a class of derivations equivalent up to the order in which they apply productions to sentential forms.

If we think of the nonterminals of the grammar as assigning grammatical categories to the strings they generate, then we can think of t as indicating the categorical structure, or *phrase structure* of the sentence whose derivations it indicates.

Chomsky notes that very many context-sensitive grammars have no sensible notion of phrase structure trees associated with their derivations. Of central interest to the field of *mathematical linguistics*, and of some interest to us as well, is the study of language families strictly between the context free and context sensitive for which a coherent notion of phrase structure exists.

2 Another way to look at it

13 Signatures A *signature* Σ is an indexed family of sets $(\Sigma_n)_{n\in\mathbb{N}}$. A symbol in Σ_n for any particular n is called an *operator of rank* n. The elements of Σ_0 are usually called *constants*.

14 Terms Given a signature Σ, the set of Σ-*terms*, T_Σ, is defined as the smallest set such that

1. $\Sigma_0 \in T_\Sigma$
2. Given $t_1, \ldots, t_n \in T_\Sigma$ and $f \in \Sigma_n$, $f(t_1, \ldots, t_n) \in T_\Sigma$.

15 Variables Let X be a countably infinite set of variables $x_1, x_2, \ldots$. For the moment assume that all variables are of rank 0. Let X_m denote the finite subset of X $x_1, x_2, \ldots, x_m$.

Let $T_\Sigma(X)$ denote the set of terms over Σ, parameterized by a set of variables X. This is defined as $T_{\Sigma\cup X}$, i.e., as if by adding the elements of X to Σ_0.

An element t of $T_\Sigma(X)$ is called *linear* if no variable of X has more than one occurrence in t.

A subset L of $T_\Sigma(X)$ is called a *term language*.

16 Algebras Given a signature Σ define a Σ-*algebra* $\mathbf{A} = \langle A, (f_\mathbf{A})_{f\in\Sigma}\rangle$, where A is an arbitrary set, called the *carrier* of $\mathbf{A}$, and for $f \in \Sigma_n$, $f_\mathbf{A}$ is an n-ary operation on the carrier set A, i.e., $f_\mathbf{A} : A^n \to A$.

17 Term Algebras From $T_\Sigma(X)$ we can induce a Σ-algebra $\mathbf{T}_\Sigma(X)$ essentially by letting terms denote themselves. That is, the operation $f_{\mathbf{T}_\Sigma(X)}$, for $f \in \Sigma_n$, is that operation which takes n terms and makes them subterms of f. That is,

$$f_{\mathbf{T}_\Sigma(X)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

18 Homomorphism Given two Σ-algebras $\mathbf{A}$ and $\mathbf{B}$, a function $h : \mathbf{A} \to \mathbf{B}$ is called a *homomorphism* just in case, for all n and all $f \in \Sigma_n$,

$$h(f_\mathbf{A}(a_1, \ldots, a_n)) = f_\mathbf{B}(h(a_1), \ldots, h(a_n))$$

for every n-tuple $(a_1, \ldots, a_n)$ in A^n.

19 Derived Operations Given the term algebra $\mathbf{T}_\Sigma$ and any (other) Σ-algebra $\mathbf{A}$, there is a unique homomorphism $h : \mathbf{T}_\Sigma \to \mathbf{A}$. The extension of this homomorphism to $\mathbf{T}_\Sigma(X)$ depends only on the way we interpret the variables in X.

Let us represent an assignment to variables in X_n of values from $\mathbf{A}$ by the tuple $\bar{a} = (a_1, \ldots, a_n)$, on the understanding that this means that x_i is assigned the value a_i. Let $h \oplus \bar{a}$ denote the (unique) extension of the homomorphism

$h : \mathbf{T}_\Sigma \to \mathbf{A}$ to a homomorphism from $\mathbf{T}_\Sigma(X_n)$ to $\mathbf{A}$, got by using $\bar{a}$ as an assignment to variables in X_n.

Any term t in $T_\Sigma(X)$ contains only a finite number of variables, so there is some n such that $t \in T_\Sigma(X_n)$. So t can be considered as defining an n-ary operation, namely that operation which maps an n-tuple $\bar{a} \in A^n$ to $(h \oplus \bar{a})(t)$. (This is just type-raising: we are taking what appear to be the arguments to t and using them to construct a function that takes t as its argument. We will see later that this is probably the proper way to interpret natural language quantifiers, for example.)

20 Substitution, again Consider now the term algebra $\mathbf{T}_\Sigma(X)$ and the (closely related) term algebra $\mathbf{T}_\Sigma(X_m)$. Suppose we have a term $t \in \mathbf{T}_\Sigma(X)$ which has at most n variables in it. Then it induces an n-ary operation over $\mathbf{T}_\Sigma(X_m)$. That is, it maps an n-tuple of trees to a new m-ary operation. The arities are different because the newly inserted trees can use more or fewer variables than the original tree.

In this case the unique extended homomorphism that we use to interpret our operator t is just exactly substitution of a term for a variable, familiar from Prolog.

We will write $t(t_1, \ldots, t_n)$ to mean (equivalently) the term t with t_i substituted for x_i or the derived term algebra operator t applied to the arguments $(t_1, \ldots, t_n)$.

21 (Context-free) Tree Grammars Now suppose we have *two* signatures Σ and $\mathcal{F}$. Call Σ the *terminals* and $\mathcal{F}$ the *nonterminals*. *Nota bene:* both of these are signatures, that is, they both contain symbols that take more than 0 arguments! So the designation "terminal" may be a bit misleading.

A *context-free tree grammar* $G = \langle \Sigma, \mathcal{F}, S, P \rangle$ is defined with Σ and $\mathcal{F}$ as above, $S \in \mathcal{F}_0$ a constant in $\mathcal{F}$, our start symbol, and P a set of productions of the form

$$F(x_1, \ldots, x_n) \longrightarrow t \qquad (n \geq 0)$$

for $F \in \mathcal{F}_n$, and $t \in T_{(\Sigma \cup \mathcal{F})(X_m)}$, that is, t is a term or tree with nodes whose labels can be drawn from either signature.

22 Derivations in a Tree Grammar Assume given a CFTG $G = \langle \Sigma, \mathcal{F}, S, P \rangle$. Suppose we have a tree t_0 in $T_{(\Sigma \cup \mathcal{F})(X_1)}$ which contains exactly one occurrence of the variable x_1 (the only variable in X_1).

Now suppose our grammar contains the production $F(x_1, \ldots, x_m) \longrightarrow t''$. Let t be a term from $T_{\Sigma \cup \mathcal{F}}$ such that

$$t = t_0(F(t_1, \ldots, t_m)$$

for some trees $t_1, \ldots, t_m$ in $T_{\Sigma \cup \mathcal{F}}$. Then we say that

$$t \Longrightarrow t'$$

(t directly derives t') just in case

$$t' = t_0(t''(t_1, \ldots, t_m))$$

In other words, we take F, thought of as a variable ranging over m-ary operations, and we replace it with t'', a derived operation of the same arity. We are careful to apply t'' to the arguments that we formerly applied F to.

23 Trees and Strings Given an ordinary vocabulary V (we will reserve Σ to refer to a signature for the time being) we can derive a corresponding signature Σ_V by setting $\Sigma_1 = V$, $\Sigma_0 = \{\epsilon\}$, and $\Sigma_n = \emptyset$ for all other n. The set of trees in T_{Σ_V} is isomorphic to the set of strings V^*, under the homomorphism interpreting each $a \in (\Sigma_V)_0$ as left concatenation by the string a in V^*, and interpreting ϵ as the empty string. In effect, the term

$$w_1(w_2(w_3(\ldots w_n(\epsilon)\ldots)))$$

is just the "vertical version" of the string

$$w_1 w_2 w_3 \ldots w_n$$

24 Tree Grammars for Regular String Languages Given this correspondence between trees and strings, we can ask what kinds of tree grammars yield the languages in $\mathcal{REG}$. The answer is the following. Let $G_1 = \langle \Sigma, \mathcal{F}, S, P \rangle$ be a context free tree grammar and let G_2 be a right linear grammar. $L(G_2)$ is therefore in $\mathcal{REG}$. Suppose that $\mathcal{F}_n = \emptyset$ for all $n > 0$. That is, $\mathcal{F}$ contains only nullary nonterminals. Furthermore, suppose that Σ contains only nullary and unary symbols, i.e., Σ_n is empty for all $n > 1$.

Then the productions in P can only be of two forms: $F \longrightarrow a(G)$ and $F \longrightarrow a$. Given our correspondence between "thin trees" and strings, these two kinds of production correspond exactly to the kinds of production allowed in a right-linear grammar.

25 Regular Tree Grammars What would be the analogue of a regular language for trees? We get trees by taking the kind of signature we derived for "rotated strings" or "thin trees", and allowing now that Σ_n may be nonempty for $n > 1$. This allows branching nodes in our trees: they are not so thin any more.

Nonetheless we retain the restriction of $\mathcal{F}$ to nullary variables.

A right linear grammar allows sentential forms that can only be expanded on the right. All the nonterminals in a sentential form are at the right edge. (In fact there is only one nonterminal in a sentential form for a right-linear grammar, so this observation is trivial.) When we move to trees by rotating our strings right 90°, the nonterminals (okay, the nonterminal) winds up at the bottom.

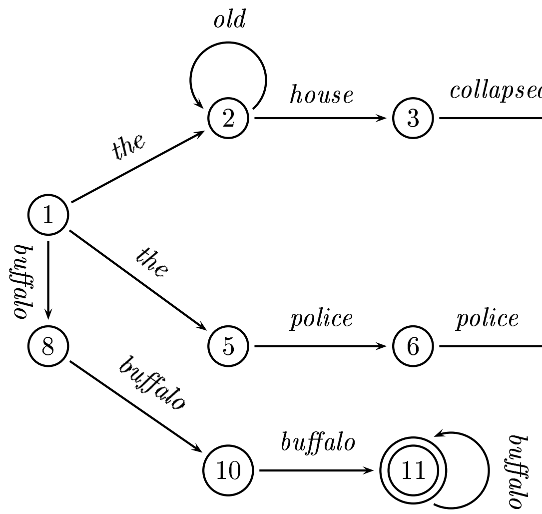
If $\mathcal{F}$ contains only nullary symbols, then these, our new nonterminals, must appear only at the bottom of a term. So a right-linear (string) grammar becomes a "bottom-linear" tree grammar.

26 Regularity for Trees It turns out that this analogy works. We can define automata that run over trees which have all the essential properties of Finite State Automata. Then we can show that regular tree languages have all the closure properties of regular string languages (in particular they are closed under intersection). We can also define an analogue of concatenation (a kind of substitution for a variable or polyadic function composition), and from that an iteration operator, so we have a notion of regular expression for trees as well.

Most importantly, all of these mechanisms describe the same family of tree languages, just as for regular string languages.

3 Too Few Examples

27 A Finite State Automaton



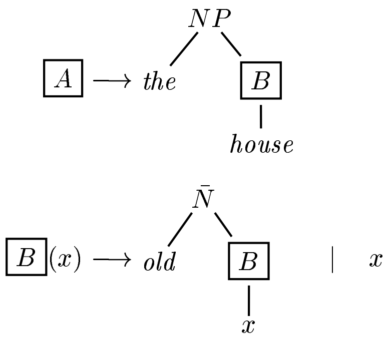
28 A Right-Linear Grammar

$$
\begin{aligned}
S &\longrightarrow \textit{the } A \\
S &\longrightarrow \textit{the } B \\
S &\longrightarrow \textit{buffalo } C \\
C &\longrightarrow \textit{buffalo } C' \\
C' &\longrightarrow \textit{buffalo } C'' \mid \textit{buffalo} \\
C'' &\longrightarrow \textit{buffalo } C'' \mid \textit{buffalo}
\end{aligned}
\qquad
\begin{aligned}
A &\longrightarrow \textit{old } A \mid \textit{house } A' \\
A' &\longrightarrow \textit{collapsed} \\
B &\longrightarrow \textit{police } B' \\
B' &\longrightarrow \textit{police } B'' \mid \textit{police} \\
B'' &\longrightarrow \textit{police } B'' \mid \textit{police}
\end{aligned}
$$

29 A Context-Free Grammar

$$\begin{aligned} S &\longrightarrow NP\ VP \\ VP &\longrightarrow V \mid V\ NP \\ NP &\longrightarrow Det\ \bar{N}\ OptRel \\ OptRel &\longrightarrow S \mid \epsilon \\ \bar{N} &\longrightarrow Adj\ \bar{N} \mid N \end{aligned} \qquad \begin{aligned} V &\longrightarrow collapsed \mid police \mid buffalo \\ Det &\longrightarrow the \\ N &\longrightarrow house \mid police \mid buffalo \\ Adj &\longrightarrow old \mid buffalo \end{aligned}$$

30 A CF Tree Grammar



References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling: Volume 1: Parsing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[2] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[3] Uwe Mönnich. Adjunction as substitution: An algebraic formulation of regular, context-free and tree adjoining languages. In Geert-Jan M. Kruijff, Glyn V. Morrill, and Richard T. Oehrle, editors, *Formal Grammar 1997: Proceedings of the Conference*, pages 169–178, 1997.

[4] Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory: Volume I Languages and Parsing.* Springer, Berlin, 1988.