

Lecture 15. Automata and Grammars I. Finite automata, regular languages, and Type 3 grammars.

Reading (for all of Lecture 15):1

1. Languages, grammars, automata: basic concepts. (Chapter 16) **Error! Bookmark not defined.**
 2. Finite automata. **Error! Bookmark not defined.**
 3. Regular languages **Error! Bookmark not defined.**
 4. Pumping Theorem for $\text{fal}'\text{s}$ 4
 5. Type 3 grammars and their relation to fsa. 4
 6. Properties of regular languages. 5
 7. Inadequacy of Type 3 grammars for natural languages. 6
- Homework 15. [repeated from last time; nothing additional now] **Error! Bookmark not defined.**

Reading (for all of Lecture 15):

Chapter 16, "Basic Concepts [of Languages, Grammars, and Automata]", of PtMW, pp. 431-452.
Chapter 17, "Finite Automata, Regular Languages, and Type 3 Grammars" of PtMW, pp. 453-484.

Erratum to PtMW p. 464: the single state q_0 in Figure 17-5 a) is a final state as well as an initial state, and should therefore be drawn with a double circle, not a single circle.

Minimal handout

Almost like the non-handout for Lecture 14, this isn't a real handout. Perhaps if things come up in class that aren't in PtMW, I'll update the handout afterwards, if some of you will lend me your class notes (as I intend to do for Lecture 11 on Mathematical Induction, for which I didn't even do a "placeholder" handout, and Lecture 14 on Infinities.)

1. Languages, grammars, automata: basic concepts. (Chapter 16)

Strings: We already have the concept of a (finite) *string over an alphabet A*. We will always assume that the set A is finite, and that we are only concerned with strings of finite length. The alphabet is sometimes called V .

Given an alphabet A , the set of all strings over A is called A^* . A^* , with the operation of concatenation, is a monoid. The identity element is the empty string, called e .

A handy aside (occurs in various examples): the unary operation "reversal" on strings: x^R .
Defined recursively on p. 433.

A nice optional exercise: prove by mathematical induction on the length of strings that for all strings x and y , $(x \wedge y)^R = y^R \wedge x^R$

Def. A *language* (over a vocabulary A) is any subset of A^* .

What is the cardinality of A^* ? What is the cardinality of the set of all languages over A ?

The set of formal devices we will consider for characterizing languages, namely formal grammars and automata, form denumerably infinite classes. So there are languages without grammars.

Grammars. Read 16.2. We will return to these.

Trees. Read 16.3. We will return to these.

Grammars and trees. Read 16.4. We will return to these.

The Chomsky hierarchy. Read 16.5. We will return to this.

Automata. Read the very brief 16.6. But we will start right in with the examples in Chapter 17.

2. Finite automata.

Terminology: *finite automaton* = *finite state automaton*. The class is sometimes called FSA, or just FA. The class of languages accepted by a fsa is called a *finite state language*. These are also the *regular languages*, but we use a separate definition, later, to define ‘regular language’, and then we prove the equivalence of the two classes. We will also define a particular class of formal grammars, the Type 3 grammars, and we can prove that the class of languages definable by Type 3 grammars is exactly the same class, the regular languages. We will thus have three independent characterizations of the same class of languages.

An automaton may be *deterministic* or *non-deterministic*. We will first define deterministic fsa, then non-deterministic, then show that for fsa (this is not true for some other classes of automata), the two subclasses of fsa are equivalent with respect to the class of languages accepted.

State diagrams. Example illustrating how dfa work. [blackboard; fig 17-2, p.456]

Formal definition:

Definition 17.1. A deterministic finite automaton (dfa) M is a 5-tuple $\langle K, \Sigma, \delta, q_0, F \rangle$, where

K is a finite set, the set of states

Σ is a finite set, the alphabet

$q_0 \in K$, the initial state

$F \subseteq K$, the final states

δ is a function from $K \times \Sigma$ into K , the transition function (or next-state function).

What makes this a *deterministic* fsa is that δ must be a function: for each state and symbol, there is exactly one transition to a next state.

Skipping in class: the formal definition of a “situation”, a binary relation “produces-in-one-move”, and the defined binary relation “produces-in-zero-or-more-moves”.

Then we can say that a dfa M *accepts* a string x in Σ^* iff M produces x in zero or more moves. The *language* $L(M)$ *accepted* by a dfa M is the set of all strings accepted by M .

Non-deterministic fa's (nfa).

Two in-principle weakenings of the requirements, and two more that are ‘optional’ but commonly included.

(i) for a given state-symbol pair, possibly more than one next state. [this is THE crucial one]

(ii) for a given state-symbol pair, possibly no next state. [this could always be modelled by adding a “dead-end state”]

(iii) allowing a transition of the form (q_i, w, q_j) where $w \in \Sigma^*$, i.e. being able to read a string of symbols in one move, not only a single symbol. And as a noteworthy subcase of that,

(iv) allowing a transition of the form (q_i, e, q_j) : changing state without reading a symbol.

Example. fig 17-3, p. 459

An input tape is *accepted* by a non-deterministic fa if there is *some* path through the state diagram which begins in the initial state, reads the entire string, and ends in a final state.

Formal definition of non-deterministic fa. Just like formal definition of dfa, except that in place of the transition function δ there is a transition relation Δ , a finite subset of $K \times \Sigma^* \times K$.

The definitions of acceptance of a string, and of accepted language, are exactly analogous to the earlier ones.

Equivalence of deterministic and non-deterministic fsa. This is a major result – it is not self-evident. The algorithm for constructing an equivalent deterministic fsa, given a non-deterministic one, is a bit complex and we won't do it; in the worst case it may give a dfa with 2^n states corresponding to a nfa with n states. (And that presupposes that we take the narrower definition of nfa, with weakenings (i) and (ii) but not (iii) or (iv).)

Why it is useful to have both notions: The deterministic fa are conceptually more straightforward; but in a given case it is often easier to construct a non-deterministic fa. Also, for some other classes of automata that we will consider, the two subclasses are not equivalent, so the notions remain important.

3. Regular languages

We define two new operations on languages, considering languages as sets of strings.

Def 17.9 (p. 462) The *concatenation* AB (or *set product*) of two sets of strings A and B . (Sometimes written $A \bullet B$) = $\{x \wedge y : x \in A, y \in B\}$

The *Kleene star* or *closure* of a set of strings A is A^* , the set of all strings formed by concatenating members of A any number of times (including zero) in any order and allowing repetitions. This is just like our existing notion A^* , except that now A can be any set of strings, not just an alphabet (i.e. a set of strings of length one). The old notion is just a special case of the new more general notion, the case where A is a set of strings of length one.

Using these two new operations on sets of strings, as well as standard set-theoretic notions, we can now define the *regular languages* recursively.

Definition 17.10 (p. 463)

Given an alphabet Σ :

1. \emptyset is a regular language.
2. For any string x in Σ^* , $\{x\}$ is a regular language.
3. If A, B are regular languages, then so is $A \cup B$.
4. If A, B are regular languages, then so is AB .
5. If A is a regular language, then so is A^* .
6. Nothing else is a regular language.

Theorem. (Kleene) A set of strings is a finite automaton language iff it is a regular language.

We can sketch one half of the proof by showing how to construct a finite state automaton corresponding to any given regular expression. (See pp. 464-468)

Steps in the proof:

- i. The empty language is a fal (finite automaton language)
- ii. The unit language for every symbol in Σ is a fal.
- iii. fal's are closed under union.
- iv. fal's are closed under concatenation.
- v. fal's are closed under the Kleene star operation.

Therefore the regular languages are included in the fal's.

[The converse proof is more complicated.]

4. Pumping Theorem for fal's

Theorem 17.2 If L is an infinite fal over alphabet Σ , then there are strings $x, y, z \in \Sigma^*$ such that $x \neq e$ and $xy^n z \in L$ for all $n \geq 0$.

Why: show that machines for infinite languages must have loops. The string y in the theorem corresponds to a string accepted during a traversal of a loop.

Note that theorem does not say 'iff'.

Main use of the theorem: in proving that some language is NOT regular.

Example: $L = \{a^n b^n \mid n \geq 0\}$

But the pumping theorem is not always useful for showing a language to be non-regular.

5. Type 3 grammars and their relation to fsa.

Review the definition of formal grammars that we skipped before.

Definition 16.2 (p.436) Let $\Sigma = V_T \cup V_N$. A *formal grammar* G is a quadruple $\langle V_T, V_N, S, R \rangle$, where V_T and V_N are finite disjoint sets, S is a distinguished member of V_N , and R is a finite set of ordered pairs in $\Sigma^* V_N \Sigma^* \times \Sigma^*$.

The ordered pairs in R are called *rules*; and $\langle \psi, \omega \rangle$ is usually written as $\psi \rightarrow \omega$ (read as " ψ rewrites as ω "). The last condition above says that the string on the left must contain at least one non-terminal symbol. Other conditions are imposed for particular classes of grammars.

Definition 16.3. Given a grammar $G = \langle V_T, V_N, S, R \rangle$, a derivation is a sequence of strings x_1, x_2, \dots, x_n ($n \geq 1$) such that $x_1 = S$ and for each x_i ($2 \leq i \leq n$), x_i is obtained from x_{i-1} by one application of some rule in R .

Definition 16.4. A grammar G *generates* a string $x \in V_T^*$ if there is a derivation x_1, \dots, x_n by G such that $x_n = x$.

Note that by this definition only strings of terminal symbols are said to be generated.

Definition 16.5. The language generated by a grammar G , denoted by $L(G)$, is the set of all strings generated by G .

The Chomsky hierarchy:

Types of grammars defined in terms of additional restrictions on the form of the rules:

Type 0: No restriction.

Type 1: Each rule is of the form $\alpha A \beta \rightarrow \alpha \psi \beta$, where $\psi \neq \epsilon$.

Type 2: Each rule is of the form $A \rightarrow \psi$. (ψ may be ϵ .)

Type 3: Each rule is of the form $A \rightarrow xB$ or $A \rightarrow x$.

Common names:

Type 0: Unrestricted rewriting systems.

Type 1: Context-sensitive grammars.

Type 2: Context-free grammars.

Type 3: Right-linear, or regular, or finite state grammars.

Correspondence of type 3 grammars and fsa's. (Construction p.473) Every type 3 language is a fal. Can also show that every fal is a type 3 language (construction p.474).

Automata viewed as either generators or acceptors.

Grammars viewed as either generators or acceptors.

6. Properties of regular languages.

Closure properties: We already know that the class of fal's is closed under union, concatenation, and Kleene star. What about intersection? Complementation?

Show complementation: if L is a fal, then $\Sigma^* - L$ is a fal. Use fsa construction. Assume we have a deterministic fal M that accepts L . We can construct a deterministic fal M' which accepts the complement of L just by interchanging final and non-final states.

Therefore fal's are also closed under intersection (why?).

Therefore the class of regular languages over any fixed alphabet is a Boolean algebra.

Decidability properties: is there an algorithm for determining ... ?

- The membership question: yes.
- The emptiness question: yes.
- Does M accept all of Σ^* ?

Problem (opt. exercise): Is there an algorithm for determining, given two machines M_1, M_2 , whether $L(M_1) \subseteq L(M_2)$? (Yes. Show it.)

Is there an algorithmic solution to the question of whether two fsa's accept the same language?

7. Inadequacy of Type 3 grammars for natural languages.

Is English a regular language? No. Use closure under intersection plus the pumping theorem. (pp 477-479)

Homework 15.

Note: for all of exercises 1-3 for Chapter 17, the alphabet must be specified as $\{0,1\}$.

PtMW, Ch 17, pp 480-484. Exercises 1-11. You don't have to do all of them, and we may not cover enough today for you to even try all of them yet, but it would be good (maybe divided into two homeworks) to try some of each 'kind', so that you have practice with the automata, the regular expressions, and the Type 3 grammars. Especially try exercises 7 and 8 so that you can get a feel for how finite state automata can form a Boolean algebra.