

# Regular Expressions

A *regular expression* is an algebraic way of defining a pattern

- We'll show that regular expressions define exactly those languages that can be accepted by a finite automaton.

**Definition:** The set of *regular expressions over  $I$*  (where  $I$  is an input set) is the least set such that:

- the symbol  $\emptyset$  is a regular expression;
- the symbol  $\lambda$  is a regular expression;
- the symbol  $\mathbf{x}$  is a regular expression if  $x \in I$ ;
- if  $A$  and  $B$  are regular expressions, then so are  $AB$ ,  $A \cup B$ , and  $A^*$ .

That is, we start with the empty set,  $\lambda$ , and elements of  $I$ , then close off under union, concatenation, and  $*$ .

Those of you familiar with the programming language Perl or Unix searches should recognize the syntax ...

Each regular expression  $E$  over  $I$  defines a subset  $I^*$ , denoted  $L(E)$  (the *language* of  $E$ ) in the obvious way:

- $L(\emptyset) = \emptyset$ ;
- $L(\lambda) = \{\lambda\}$ ;
- $L(\mathbf{x}) = \{x\}$ ;
- $L(AB) = L(A)L(B)$ ;
- $L(A \cup B) = L(A) \cup L(B)$ ;
- $L(A^*) = L(A)^*$ .

### Examples:

- What's  $0^*10^*10^*$ ?
- What's  $(0^*10^*10^*)^n$ ?  $(0^*10^*10^*)^*$ ?
- $(0^*10^*10^*)^*$  is the language accepted by the parity automaton!
- If  $\Sigma = \{a, \dots, z, A, \dots, Z, 0, \dots, 9\} \cup Punctuation$ , what is  $\Sigma^*Halpern\Sigma^*$ ?
  - *Punctuation* consists of the punctuation symbols (comma, period, etc.)
  - $\Sigma$  is an abbreviation of  $a \cup b \cup \dots$  (the union of the symbols in  $\Sigma$ )

Can you define an automaton that accepts exactly the strings in  $\Sigma^*Halpern\Sigma^*$ ?

- How many states would you need?

What language is represented by the automaton in the original example:

- $((10)^*0^*((110) \cup (111))^*)^*$

What language is accepted by the following three automata (Rosen, p. 807, Figure 2)?

$1^*$

$1 \cup 01$

$0^* \cup 0^*10(0 \cup 1)^*$

# Nondeterministic Finite Automata

So far we've consider *deterministic* finite automata (DFA)

- what happens in a state is completely determined by the input. symbol read

*Nondeterministic* finite automata allow several possible next states when an input is read.

Formally, a nonsterministic finite automaton is a tuple  $M = (S, I, f, s_0, F)$ . All the components are just like a DFA, except now  $f : S \times I \rightarrow 2^S$  (before,  $f : S \times I \rightarrow S$ ).

- if  $s' \in f(s, i)$ , then  $s'$  is a possible next state if the machines is in state  $s$  and sees input  $i$ .

We can still use a graph to represent an NFA. There might be several edges coming out of a state labeled by  $i \in I$ . In the example below (Rosen, p. 812; Figure 7), there are two edges coming out of  $s_0$  labeled 0.

- can either stay in  $s_0$  or move to  $s_2$

- An NFA  $M$  *accepts* (or *recognizes*) a string  $x$  if it is possible to get to a final state from the start state with input  $x$ .
- The language  $L$  is accepted by an NFA  $M$  consists of all strings accepted by  $M$ .

What language is accepted by the NFA above?

$$0^* \cup 0^*01 \cup 0^*11$$

Problem: Write an automaton that accepts a string if it contains “man” as a substring. Here’s the obvious choice:

This doesn’t quite work: For example, it won’t accept “command”.

- We can correct the problem using nondeterminism.

# Equivalence of Automata

Every DFA is an NFA, but not every NFA is a DFA.

- Do we gain extra power from nondeterminism?
  - Are there languages that are accepted by an NFA that can't be accepted by a DFA?
  - Somewhat surprising answer: NO!

Define two automata to be *equivalent* if they accept the same language.

Examples:

**Theorem:** Every nondeterministic finite automaton is equivalent to some deterministic finite automaton.

**Proof:** Given an NFA  $M = (S, I, f, s_0, F)$ , let  $M' = (2^S, I, f', \{s_0\}, F')$ , where

- $f'(A, i) = \{t : t \in f(s, i) \text{ for some } s \in A\} \in 2^S$ 
  - $f : 2^S \times \Sigma \rightarrow 2^S$
- $F' = \{A : A \cap F \neq \emptyset\}$

Thus,

- the states in  $M'$  are subsets of states in  $M$ ;
- the final states in  $M'$  are the sets which contain a final state in  $M$ ;
- in state  $A$ , given input  $i$ , the next state consists of all possible next states from an element in  $A$ .

$M'$  is *deterministic*.

- This is called the *subset* construction.
- The states in  $M'$  are subsets of states in  $M$ .

We want to show that  $M$  accepts  $x$  iff  $M'$  accepts  $x$ .

- Let  $x = x_1 \dots x_k$ .
- If  $M$  accepts  $x$ , then there is a sequence of states  $s_0, \dots, s_k$  such that  $s_k \in F$  and  $s_{i+1} \in f(s_i, x_i)$ .
  - That's what it means for an NFA  $M$  to accept  $x$
  - $s_0, \dots, s_k$  is a possible sequence of states that  $M$  goes through on input  $x$ 
    - \* It's only one possible sequence:  $M$  is an NFA
- Define  $A_0, \dots, A_k$  inductively:  
 $A_0 = \{s_0\}$  and  $A_{i+1} = f'(A_i, x_i)$ .
  - $A_0, \dots, A_k$  is the sequence of states that  $M'$  goes through on input  $x$ .
    - \* Remember: a state in  $M'$  is a set of states in  $M$ .
    - \*  $M'$  is deterministic: this sequence is unique.
  - An easy induction shows that  $s_i \in A_i$ .
  - Therefore  $s_k \in A_k$ , so  $A_k \cap F \neq \emptyset$ .
  - Conclusion:  $A_k \in F'$ , so  $M'$  accepts  $x$ .



For the converse, suppose that  $M'$  accepts  $x$

- Let  $A_0, \dots, A_k$  be the sequence of states that  $M'$  goes through on input  $x$ .
- Since  $A_k \cap F \neq \emptyset$ , there is some  $t_k \in A_k \cap F$ .
- By induction, if  $1 \leq j \leq k$ , can find  $t_{k-j} \in A_{k-j}$  such that  $t_{k-j+1} \in f(t_{k-j}, x_{k-j})$ .
- Since  $A_0 = \{s_0\}$ , we must have  $s_0 = t_0$ .
- Thus,  $t_0 \dots t_k$  is an “accepting path” for  $x$  in  $M$
- Conclusion:  $M$  accepts  $x$

## Notes:

- Michael Rabin and Dana Scott won a Turing award for defining NFAs and showing they are equivalent to DFAs
- This construction blows up the number of states:
  - $|S'| = 2^{|S|}$
  - Sometimes you can do better; in general, you can't

Example: