Formal Languages

We'll use the *English* language as a running example.

Definitions.

- A *string* is a *finite set* of *symbols*, where each *symbol* belongs to an *alphabet* denoted by Σ.
- The *set* of *all strings* that can be constructed from an *alphabet* Σ is Σ^*.
- If x, y are two strings of *lengths* $|x|$ and $|y|$, then:
  - xy or $x \circ y$ is the *concatenation* of x and y, so the length, $|xy| = |x| + |y|$
  - $(x)^R$ is the *reversal* of x
  - the k^{th}-*power* of x is

$$x^k = \begin{cases} \epsilon & \text{if } k = 0 \\ x^{k-1} \circ x, & \text{if } k > 0 \end{cases}$$

  - *equal, substring, prefix, suffix* are defined in the expected ways.
  - Note that the language $\emptyset$ is *not* the same language as ϵ.

Examples.

Operations on Languages

Suppose that L_E is the *English* language and that L_F is the *French* language over an alphabet Σ.

- **Complementation**: $\overline{L} = \Sigma^* - L$

  $\overline{L}_E$ is the set of all words

- **Union:** $L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$

  $L_E \cup L_F$ is the set

- **Intersection:** $L_1 \cap L_2 = \{x : x \in L_1 \text{ and } x \in L_2\}$

  $L_E \cap L_F$ is the set

- **Concatenation:** $L_1 \circ L_2$ is the set of all strings xy such that $x \in L_1$ and $y \in L_2$

  Q: What is an example of a string in $L_E \circ L_F$?

  Q: What if L_E or L_F is $\emptyset$? What is $L_E \circ L_F$?

- **Kleene star:** L^*. Also called the *Kleene Closure* of L and is the *concatenation* of *zero* or *more strings* in L.

  Recursive Definition

  - **Base Case**: $\epsilon \in L$
  - **Induction Step**: If $x \in L^*$ and $y \in L$ then $xy \in L^*$

- **Language Exponentiation** Repeated *concatenation* of a language L.

$$L^k = \begin{cases} \{\epsilon\} & \text{if } k = 0 \\ L^{k-1} \circ L, & \text{if } k > 0 \end{cases}$$

- **Reversal** The language *Rev(L)* is the language that results from *reversing all strings in L*.

Q: How do we *define* the strings that belong to a *language* such as *English, French, Java, arithmetic,* etc.

Example: For the *language of arithmetic*, $\mathcal{LA}$:

Define $\Sigma = \{\mathbb{N}\} \cup \{+, -, =, (,)\}$ then

$$\text{“)((2(+4(=”} \in \Sigma^*$$

but

$$\text{“)((2(+4(=”} \notin \mathcal{LA}.$$

Regular Expressions

A *regular expression* over an alphabet Σ consists of

1. *Symbols* in the *alphabet*
2. The symbols $\{+, (,), ^*\}$ where $+$ means OR and * means *zero* or *more times*.

Recursive Definition.

Let the set $\mathcal{RE}$ of *ALL regular expressions*, be the smallest set such that:

- **Basis**: $\emptyset, \epsilon, a \in RE, \forall a \in \Sigma$
- **Inductive Step**: if R and S are *regular expressions* $\in \mathcal{RE}$, then so are: $(R + S), (RS), R^*$

Examples: Let $\Sigma = \{0, 1\}$:

Regular Expression	Corresponding Language
$(0 + 1)^*$	all binary strings.
$((0 + 1)(0 + 1)^*)$	all non-empty binary str.
$((0 + 1)(0 + 1))^*$	all even lengthed binary strings.
$\epsilon + 0 + 0(0 + 1)^*0$	all strings not starting or ending with 1.
$11(0 + 11)^*$	all strings with 1's in pairs and starting with 11.

Relating Regular Expressions to Languages

Let $\mathcal{L}(\mathcal{R})$ represent the *language* constructed by the *regular expression* R.

We define $\mathcal{L}(\mathcal{R})$ *inductively* as follows:

Base Case:

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\epsilon) = \{\epsilon\}$
- For any $a \in \Sigma$, $\mathcal{L}(a) = \{a\}$

Induction Step: If R is a *regular expression*, then by definition of R,

- $R = ST$, or
- $R = S + T$, or
- $R = S^*$

where S and T are *regular expressions* and by *induction*, $\mathcal{L}(\mathcal{S})$ and $\mathcal{L}(\mathcal{T})$ have been defined.

We can define the language denoted by R, ie., $\mathcal{L}(\mathcal{R})$ as follows:

- $\mathcal{L}((\mathcal{S}+\mathcal{T}))$ = $L(S)$ union $L(T)$
- $\mathcal{L}((\mathcal{S}\mathcal{T}))$ = $L(S) \circ L(T)$
- $\mathcal{L}(\mathcal{S}^*)$ = $(L(S))^*$

Q: Why is this definition important?

Use these definitions when proving languages are equivalent to each other.

Example

Q: What is a *regular expression* R_A to denote the language of strings consisting of only an *even number* of a's?

e.g., $aa, aaaa, aaaaaaaa$ etc.

L $(aa)^*$

Q: What is a *regular expression* $\mathcal{R}_\mathcal{B}$ for the language of strings consisting of *1* or *more triples* of b's? e.g., $bbb, bbbbbb, bbbbbbbbb$.

$(bbb)(bbb)^*$

Q: What is a regular expression, $\mathcal{R}_{\mathcal{AB}}$, for the language of strings consisting of an *even number* of a's *sandwiched* between 1 or more *triples* of b?

eg., $bbbaabbb$, or $bbbaaaaaabbb$ $R_B R_A R_B$.

Equivalence. We say that two regular expressions R and S are *equivalent* if they *describe* the *same language*.

In other words, if $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{S})$ for two regular expressions R and S then $R = S$.

Examples.

- Are R and S *equivalent*?

$$R = a^*(ba^*ba^*)^* \text{ and } S = a^*(ba^*b)^*a^*$$

bbaabb bb

Q: Why? bbaabb $\in R$ $\notin S$.

- Are $R = (a(a+b)^*)$ and $S = (a(a+b))^*$ *equivalent*?

$\varepsilon \in R$, $\varepsilon \in S$.

Regular Expression Equivalences

There exist *equivalence axioms* for *regular expressions* that are very similar to those for *predicate/propositional logic*.

Equivalences for Regular Expressions R, S, T are R.E.

- Commutativity of union: $R + S \equiv S + R$
- Associativity of union: $(R + S) + T \equiv R + (S + T)$
- Associativity of concatenation: $R(ST) \equiv (RS)T$
- Left distributivity: $R(S + T) \equiv RS + RT$
- Right distributivity: $(S + T)R \equiv SR + TR$
- Identity of Union: $R + \emptyset \equiv R$
- Identity of Concatenation: $R\varepsilon \equiv R$
- Annihilator for concatenation: $R\emptyset \equiv \emptyset \equiv \emptyset R$
- Idempotence of Kleene star: $R^{**} \equiv R^*$

Theorem (Substitution) *If two substrings R and R' are equivalent then if R is a substring of S then replacing R by R' constructs a new regular expression equivalent to S.*

Equivalent Regular Expressions

Q: How can we determine whether two *regular expressions* denote the *same language*?

→ use reg. expn equivalences to show they are the same

→ find a string in one and not the other

Examples.

Prove that

$$(0110 + 01)(10)^* \equiv 01(10)^*$$

Proof.

$(0110 + 01)(10)^* \equiv (0110 + 01\varepsilon)(10)^*$ by id. of concat.
$\equiv (01(10 + \varepsilon))(10)^*$ left. dist.
$\equiv (01)((10+\varepsilon)(10)^*)$ assoc. of concat
$\equiv 01(10(10)^* + \varepsilon(10)^*)$ right dist.
$\equiv 01(10(10)^* + (10)^*)$ id of concat.
$\equiv 01(10)^*$ by substitution of equiv.
$\equiv$ R.E.

Notice that $L(10(10)^*) \subseteq L((10)^*)$

Another Example.

Prove that R denotes the *language* L of all strings that contain an *even number* of 0s.

$$R = 1^*(01^*01^*)^*$$

Equivalently,

$$x \in L \Leftrightarrow x \in \mathcal{L}(R)$$

Proof.

~~($\Rightarrow$)~~ ($\Leftarrow$)

- Let $x \in \mathcal{L}(R)$.
- Then $x \in \mathcal{L}(1^*(01^*01^*)^*) = \mathcal{L}(1^*)(\mathcal{L}(01^*)\mathcal{L}(01^*))^*$
- Let $x = y(zw)^*$ then $y \in L(1^*)$, $z \in L(01^*)$, $w \in L(01^*)$
- Therefore, y has only 1's.
- Therefore, w has exactly one 0.
- Therefore, z has exactly one 0.
- So, $x = y(zw)^*$ has a multiple of two 0's.

$x \in L$

($\Leftarrow$)($\Rightarrow$)

- Suppose that x is an *arbitrary* string in L.
- $\Rightarrow$ x has an *even* number of 0s. Denote by $2k$ for some $k \in \mathbb{N}$.
- How can we rewrite x consisting of 0s and 1s?
- Let $x = y_0, y_1, y_2, \ldots, y_k$, so $y_0 = 1 \cdots 1 \in \mathcal{L}(1^*)$

each $y_i = (01\cdots1)(01\cdots1) = 01^n01^m \in \mathcal{L}(01^*)\mathcal{L}(01^*)$, $1 \le i \le k$, y_i

- So $x = y_0 y_1 \ldots y_k \in \mathcal{L}(1^*)(\mathcal{L}(01^*01^*))^* = \mathcal{L}(1(01^*01^*)^*)$.

(y_0, y_i)

Q: Can *every* possible type of string be *represented* by a *regular expression*?

To answer this, we turn to *Finite State Machines*.

String Matching and Finite State Machines

- Given *source code* (say in Java)
- Find the *comments* – may need to remove comments for *software transformations*

```
public class QuickSort {
    private static long comparisons = 0;
    private static long exchanges   = 0;

   /*****************************************************************
    *  Quicksort code from Sedgewick 7.1, 7.2.
    *****************************************************************/
    public static void quicksort(double[] a) {
        shuffle(a);                        // to guard against worst-case
        quicksort(a, 0, a.length - 1);
    }
    public static void quicksort(double[] a, int left, int right) {
        if (right <= left) return;
        int i = partition(a, left, right);
        quicksort(a, left, i-1);
        quicksort(a, i+1, right);
    }

    private static int partition(double[] a, int left, int right) {
        int i = left - 1;
        int j = right;
        while (true) {
            while (less(a[++i], a[right]))      // find item on left to swap
                ;                               // a[right] acts as sentinel
            while (less(a[right], a[--j]))      // find item on right to swap
                if (j == left) break;           // don't go out-of-bounds
            if (i >= j) break;                  // check if pointers cross
            exch(a, i, j);                      // swap two elements into place
        }
        exch(a, i, right);                      // swap with partition element
        return i;
    }
```

Q. What *patterns* are we looking for?

// text \n /* text */

Q. What do we know if we see a / followed by a

* in a comment

/ in a comment

text not in a comment

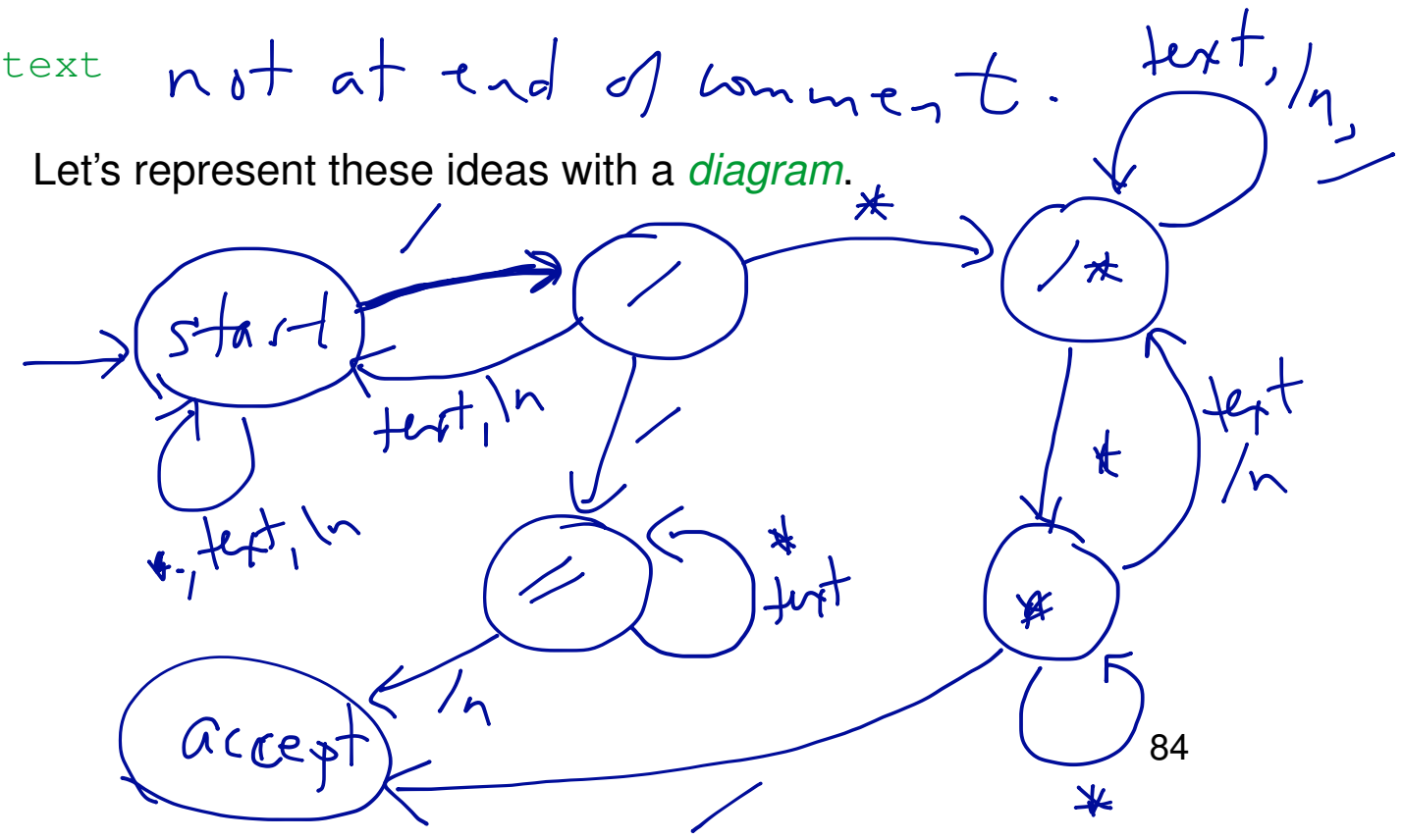
Q. What do we know if we see /* followed by a

* might be at the end of the comment if next char is a /

/ not at end

text not at end of comment.

Let's represent these ideas with a *diagram*.



Deterministic Finite State Automata (DFSA or DFA)

A *DFA* consists of:

- Q. the set of states
- Σ. an alphabet from with strings come from.
- $s \in Q$. start state
- $F \subseteq Q$. final or accepting states.
- δ. $Q \times \Sigma \rightarrow Q$ transition function which takes a state and input and outputs a state.

Comment Example.

- $Q = \{start, /, //, /*, *, accept\}$.
- $\Sigma = \{text, \backslash n, /, *\}$
- $s =$ start
- $F = \{accept\}$
- δ

Example cont...

δ(state, input)	/	*	text	\nl
start	/	start	start	start
/	//	/*	start	start
//<-	//	//	//	accept
/*	/*	*	/*	/*
accept	/	accept		
*	accept	*	/*	/*

Q: What if we want to know which state the input "**//" ends at if we *begin* at *start*?

Two Options.

1. Compute: $\delta(\delta(\delta(\delta(\delta(start, *), *), /), /) = //$
2. Define δ^*. $= \delta^*(start, "**//") = //$

Formal definition of $\delta^*(q, x)$ (reading left to right):

$$\delta^*(q, x) = \begin{cases} q & if \quad x = \epsilon \\ \delta(\delta^*(q, z), a) & if \quad x = za, a \in \Sigma, z \in \Sigma^* \end{cases}$$

Exercise: Rewrite δ^* as for reading a string from right to left.

Regular Expressions and DFA

- The set of strings *accepted* by an *automaton* defines a *language*.
- For *automaton* M the language M accepts is $\mathcal{L}(M)$.
- Given *regular expression* R, find M such that

$$\mathcal{L}(R) = \mathcal{L}(M).$$

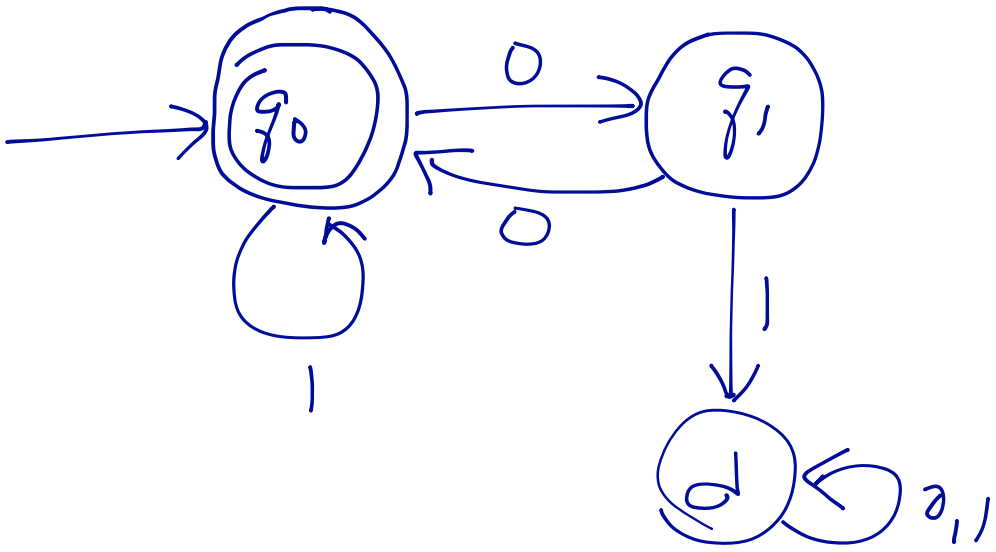
Examples.

Let *regular expression* $R_1 = (1 + 00)^*$.

Q. Which strings belong to $\mathcal{L}(R_1)$?

$L(R_1) = \{x \in \{0,1\}^* \mid$ all 0's are in pairs$\}$

Q: What is a *DFA* M_1 such that $\mathcal{L}(M_1) = \mathcal{L}(R_1)$?



DFSA Conventions

- Strings ending at a *final state* are *accepted* (if we want to accept/reject).
- Drop *dead* states.
- Group *elements* that go from and to the *same states*.

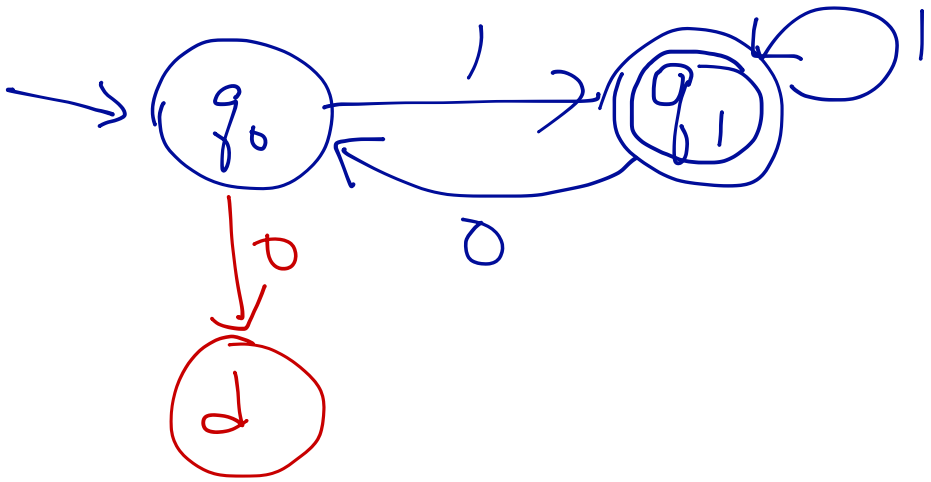
Examples cont.

Let *regular expression* $R_2 = 1(1 + (01))^*$.

Q. Which strings belong to $\mathcal{L}(R_2)$?

every 0 is sandwiched 1's.

Q: What is a *DFA* M_2 such that $\mathcal{L}(M_2) = \mathcal{L}(R_2)$?



$$\delta: \quad \delta(q_0, 0) = d \quad \delta(q_0, 1) = q_1, \delta(q_1, 0) = q_0$$

$$\delta(q_1, 1) = q_1, \quad \delta(q_2, 0) = \quad \delta(q_2, 1) =$$

$$\delta(d_1, 0 \; or \; 1) = d$$

Q: How do we know that our *machine* M is *correct*?

We can show this by proving that $\delta^*(q_0, x)$ *only accepts those strings in* $\mathcal{L}(R_2)$.

Q: What might be a *good* way to do this?

Proving a DFA is Correct

Q: What should we do *induction* on?

length or structure of the string.

Q: What should our $S(x)$ include?

a definition for δ^* and a description of the strings accepted by it. So "x".